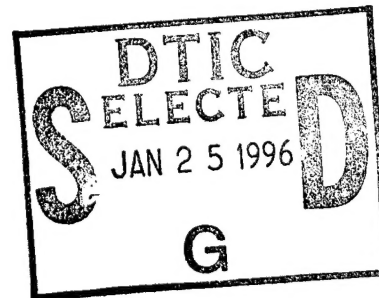


# NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



## THESIS



### SENSOR BASED NAVIGATION AND LOCALIZATION METHODS FOR AUTONOMOUS UNDERWATER VEHICLES

by

Kevin D. Conowitch

June, 1995

Thesis Advisor:

Roberto Cristi

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19960118 024

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis		
4. TITLE AND SUBTITLE SENSOR BASED NAVIGATION AND LOCALIZATION METHODS FOR AUTONOMOUS UNDERWATER VEHICLES		5. FUNDING NUMBERS		
6. AUTHOR(S) Kevin D. Conowitch				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (maximum 200 words) An algorithm designed to navigate an Autonomous Underwater Vehicle (AUV) within a charted environment is presented. The algorithm processes sensor inputs from the AUV high resolution scanning sonar, compass and velocimeter. The operating environment is modeled with a suitable three-dimensional potential function and its gradient which form an attractive field. This algorithm provides performance comparable to the Kalman Filter with the advantage of reduced computational requirements facilitated by pre-computation and table look-up of correction factors. Applications of data smoothing filters and sliding mode theory are also investigated. The applicability and robustness of this approach are demonstrated with actual test data obtained with the NPS <i>Phoenix</i> submersible and extended simulation of complex environments including unmodeled obstacles.				
14. SUBJECT TERMS Autonomous Underwater Vehicle, estimation, navigation, potential functions, sliding mode observer		15. NUMBER OF PAGES 104		
		16. PRICE CODE		
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	



Approved for public release; distribution is unlimited.

**SENSOR BASED NAVIGATION AND  
LOCALIZATION METHODS FOR  
AUTONOMOUS UNDERWATER VEHICLES**

Kevin D. Conowitch  
Lieutenant Commander, United States Navy  
B.S., University of California at Los Angeles, 1984

Submitted in partial fulfillment  
of the requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**

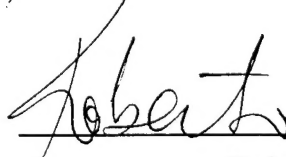
**June 1995**

Author:



Kevin D. Conowitch


Approved by:



Roberto Cristi, Thesis Advisor



Robert G. Hutchins, Second Reader



Michael A. Morgan, Chairman

Department of Electrical and Computer Engineering



## ABSTRACT

An algorithm designed to navigate an Autonomous Underwater Vehicle (AUV) within a charted environment is presented. The algorithm processes sensor inputs from the AUV high resolution scanning sonar, compass and velocimeter. The operating environment is modeled with a suitable three-dimensional potential function and its gradient which form an attractive field. This algorithm provides performance comparable to the Kalman Filter with the advantage of reduced computational requirements facilitated by pre-computation and table look-up of correction factors. Applications of data smoothing filters and sliding mode theory are also investigated. The applicability and robustness of this approach are demonstrated with actual test data obtained with the NPS *Phoenix* submersible and extended simulation of complex environments including unmodeled obstacles.

Accession For	
NTIS	CRA&I <input checked="checked" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification .....	
By .....	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



## TABLE OF CONTENTS

I. INTRODUCTION .....	1
A. GENERAL .....	1
B. EMPHASIS OF RESEARCH .....	1
C. METHOD OF APPROACH .....	2
II. BACKGROUND .....	5
A. AUV MOTION MODEL .....	5
B. DESCRIPTION OF THE ENVIRONMENT .....	6
C. INTERACTION BETWEEN MODELS .....	8
D. CONVERGENCE .....	10
E. IMPORTANCE OF PARAMETER SELECTION .....	14
III. APPLICATION OF POTENTIAL FUNCTIONS .....	17
A. ENVIRONMENTS WITH BORDERS .....	17
B. SIMULATION RESULTS .....	17
C. SIMULATIONS WITH ACTUAL DATA .....	19
D. INCORPORATION OF OBSTACLES IN THE OPERATING ENVIRONMENT .....	21
1. Unmodeled Obstacles .....	21
2. Obstacles Modeled in Potential Function .....	24
IV. APPLICATIONS IN COMPLEX ENVIRONMENTS .....	27
A. ENVIRONMENTS WITHOUT BORDERS .....	27
B. SIMULATION RESULTS .....	27
1. Accounting for Effects of Currents .....	27
C. COMPLEX ENVIRONMENTS .....	31
D. COMPLEX SIMULATION RESULTS .....	33



V. APPLICATION OF SLIDING MODE THEORY .....	37
A. SLIDING OBSERVERS FOR NONLINEAR SYSTEMS .....	37
1. One-Dimensional Example .....	37
2. Verification of Convergence .....	39
VI. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS .....	43
A. SUMMARY .....	43
B. CONCLUSIONS .....	43
C. RECOMMENDATIONS .....	44
APPENDIX A. PARAMETER ESTIMATION .....	45
APPENDIX B. SONAR CHARACTERISTICS .....	51
APPENDIX C. MATLAB AND SIMULINK FILES .....	53
A. GENERAL .....	53
B. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER II ....	53
C. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER III ....	57
D. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER IV ....	71
E. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER V ....	79
F. AUV SIMULINK MODEL .....	84
APPENDIX D. SMOOTHING FILTER DESIGN .....	87
LIST OF REFERENCES .....	91
INITIAL DISTRIBUTION LIST .....	93

## **ACKNOWLEDGEMENTS**

This thesis is dedicated to the memory of my father, Joseph H. Conowitch (1936-1993) whose guidance and encouragement continue to inspire me to make him proud.

I would also like to acknowledge my family Charmaine, Celeste and Roxanne for their unending love and support in all my endeavors.

Lastly, my thanks go to Roberto Cristi, and Robert (Gary) Hutchins, two professors who have made my time at the Naval Postgraduate School an enjoyable, challenging, but most of all a rewarding experience.

## **I. INTRODUCTION**

### **A. GENERAL**

Navigation is the process of safely directing the movements of a vehicle from one point to another. Localization is the process of accurately determining the vehicle's position in space at any given moment in time. The investigation and development of methods to accomplish both of these related processes is the topic of this thesis.

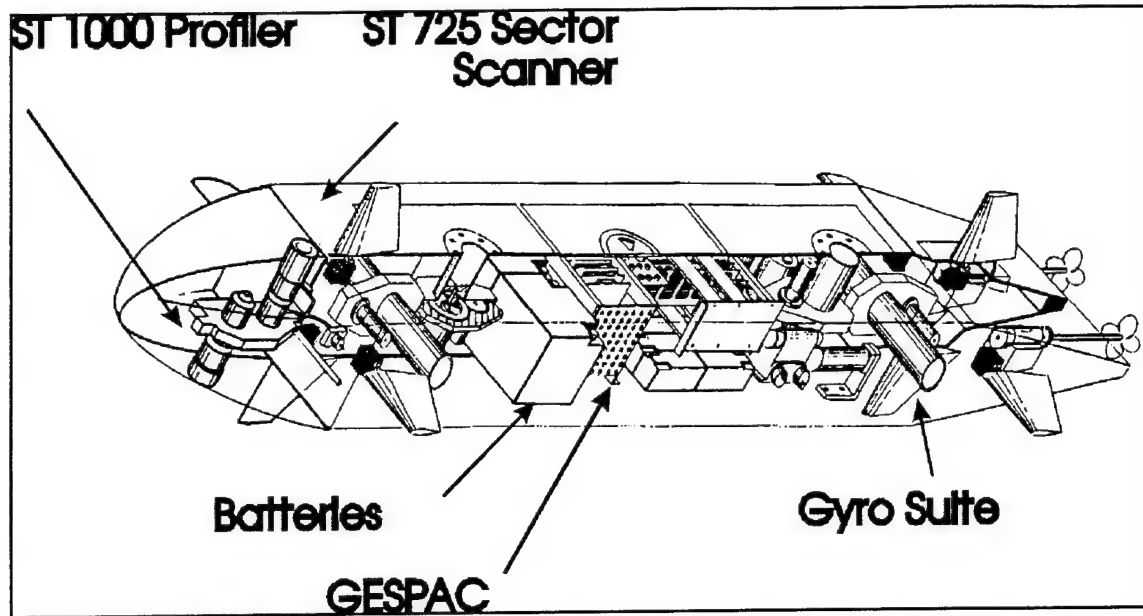
Autonomous Underwater Vehicles (AUV) are a subclass of Unmanned Underwater Vehicles (UUV) which operate under the control of systems which are an integral part of the vehicle itself. This is in direct contrast to tethered and remotely operated vehicles which require an interface and a communications link with an operator stationed in a separate location.

The advantages gained by autonomy from an external control system are obtained at the cost of a more complex on-board navigation system. Highly accurate and reliable inertial navigation systems are readily available and ideally suited for underwater navigation, but the size, weight and associated data processing characteristics of these systems preclude their practical use on small vehicles in the class of AUVs about which we are concerned.

AUVs have found increasing use in areas such as cable-laying, minefield surveillance, petroleum drilling, under-ice surveillance, and underwater construction and inspection [Ref. 1]. An AUV can be particularly useful in an environment which is hazardous or difficult to operate in for extended periods of time for human beings.

### **B. EMPHASIS OF RESEARCH**

This thesis is concerned with the navigation of a small class of AUV such as the NPS *Phoenix*. A diagram of this vehicle is shown in Figure 1.1. The methods developed will utilize the information provided by installed sensors which include a compass for determining vehicle heading, a velocimeter or log to measure vehicle speed relative to the water, a fathometer for providing depth-sensing information, and a high-frequency scanning



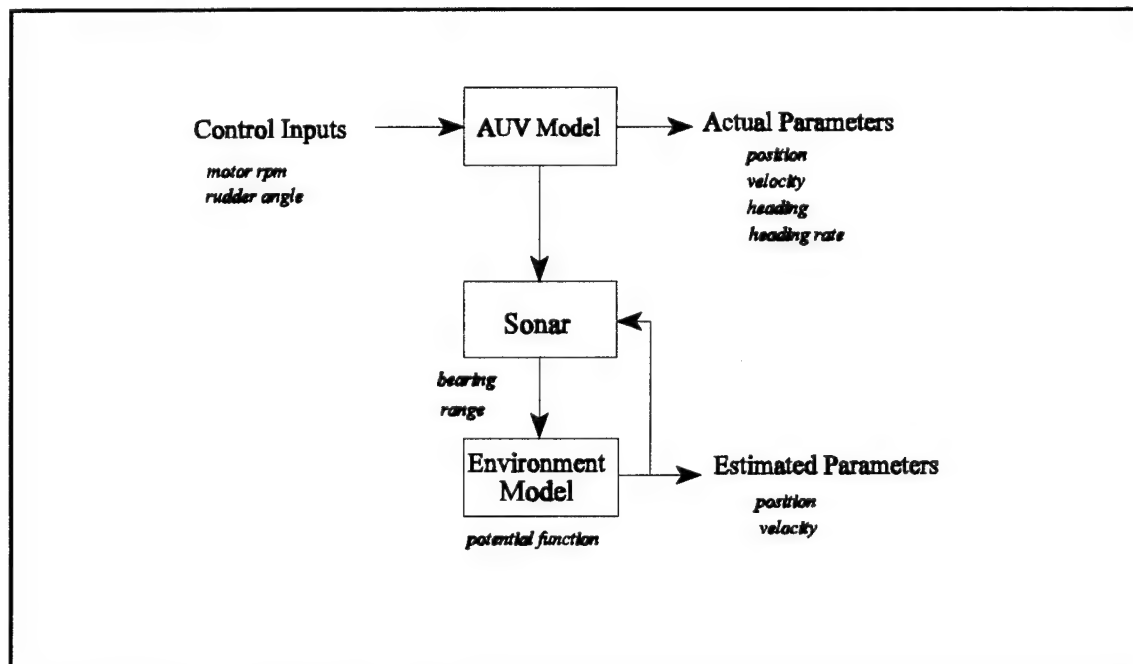
**Figure 1.1** NPS *Phoenix* AUV

sonar providing range and bearing measurements to objects in the vehicle's operating environment. This is the minimal set of required instrumentation to facilitate navigation of an underwater vehicle in three dimensions. It is the aim of this thesis to develop an effective algorithm which estimates the AUV position without utilizing a Kalman Filter. Ease of implementation, numerical stability and satisfactory performance are driving factors behind the design of the algorithm as well as a conscious effort to avoid the complexity and possible numerical instabilities associated with the Kalman Filter.

### **C. METHOD OF APPROACH**

Solving the short-range navigation problem for the AUV requires an algorithm which consists of two main parts. Firstly, a non-linear state-space model for motion of the AUV is developed which provides the AUV position and associated rates based on the system control inputs. Secondly, a model of the environment is constructed which exhibits several desirable properties. The function which defines the environment allows the precomputation of navigation correction factors for any region of operation within the mapped environment.

The onboard sonar provides the necessary interaction between the AUV and the environment models. The sonar range and bearing measurements are added vectorially to the estimated position of the AUV and the corresponding navigation correction factor is determined to adjust the estimated position closer to the actual position of the vehicle. This process is conducted in an iterative fashion providing constant position correction consistent with the availability of sonar data. This results in a simple Least Mean Squares (LMS) algorithm, which basically is a simplified Kalman Filter. A block diagram of the overall system is presented in Figure 1.2.



**Figure 1.2 System Block Diagram**

This thesis is organized in six parts. Chapter II provides the theory behind the design of both the vehicle and environmental models, as well as the interaction between the two provided by the sonar system. Chapter III applies the developed algorithm to closed-border environments such as the test pool utilized for data collection with the NPS *Phoenix*. Chapter IV demonstrates the applicability of the algorithm to more realistic scenarios of open environments and complex environment models composed of multiple geometric

shapes (or primitives). Chapter V investigates the application of sliding mode theory. Finally, Chapter VI summarizes the results and conclusions of this study and provides several recommendations for follow-on research.

## II. BACKGROUND

### A. AUV MOTION MODEL

The dynamics of the vehicle are modeled in a state-space representation. For the purpose of this analysis, the model will be restricted to planar motion of the AUV in a cartesian coordinate system with the origin as the bottom left or southwest corner of the operating environment. Defining the state of the vehicle in terms of absolute position  $x, y$  and velocity, heading and yaw rate ( $v, \theta, \dot{\theta}$ ) results in a five-dimensional state vector. This leads to a dynamic model of the form

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{w}) \quad (2.1)$$

with  $\mathbf{x} = [x \ y \ v \ \theta \ \dot{\theta}]^T \in \mathbb{R}^5$  the state vector,  $\mathbf{u} = [u_1 \ u_2]^T \in \mathbb{R}^2$  the vector of command inputs for motor rpm and rudder angle, and  $\mathbf{w}$  disturbances which account for modeling errors between Equation (2.1) and the actual physical dynamics of the vehicle.

The differential equations which comprise the function  $f$  are

$$\left\{ \begin{array}{l} \dot{x} = v \cos \theta \\ \dot{y} = v \sin \theta \\ \dot{v} = -a_1 v + b_1 u_1 \\ \ddot{\theta} = -a_2 \dot{\theta} + b_2 u_2 \end{array} \right. \quad (2.2)$$

where  $a_1, a_2, b_1$ , and  $b_2$  are constants determined by the vehicle dynamics.

A recursive least squares (RLS) parameter estimation applied to vehicle trajectory data obtained with the NPS *Phoenix* AUV to estimate the values of the dynamic parameters  $a_1, a_2, b_1$ , and  $b_2$  results in the following non-linear state-space model with  $\mathbf{w}$  corresponding to the modeling errors :

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & \cos \theta_0 & 0 & 0 \\ 0 & 0 & \sin \theta_0 & 0 & 0 \\ 0 & 0 & -508.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1.667 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u} + \mathbf{w} \quad (2.3)$$

A summary of the methodology and results of the parameter estimation are included in Appendix A. This state-space model produces parameters in units of feet (ft) for position, feet/second (fps) for velocity, degrees for heading (referenced to the x-axis), and degrees/second (dps) for heading rate. As mentioned previously, inputs are motor revolutions/minute (rpm) and rudder angle in degrees (positive values indicate left while negative values correspond to right rudder commands).

This model can be utilized in two ways. Firstly, it is used to generate simulated motion data for verifying the validity of the navigation algorithm. Secondly, it can be used as the basis of an algorithm which solves for the required control trajectory which causes the vehicle to conform to a desired track or path through the environment.

## B. DESCRIPTION OF THE ENVIRONMENT

The operating environment of the AUV is modeled by a function  $V$  which associates a potential field to any point in the environment, with the assumption that all acoustic reflective surfaces are equipotential. This is analogous to the electromagnetics model of electric charges at rest on a conducting surface, where the electric potential is constant and assumed to be zero. A potential function of this type then exhibits the following desirable properties:

- $V(\mathbf{x}) = 0$  at any point on the boundary or reflecting surface of the environment
- $V(\mathbf{x})$  is continuous, differentiable and uniformly bounded at any point.



- Given any point  $\mathbf{x}$  on the reflective surface (such that  $V(\mathbf{x}) = 0$ ), for any small perturbation  $\delta$  we can write  $\delta^T \nabla V(\mathbf{x} + \delta) \leq 0$  (with  $\nabla$  indicating the gradient operator).

The last property simply states that the vector field formed from the spatial gradient is attractive toward the reflective surface, and there is no component tangent to the surface which would violate the steady-state assumption of the field.

To illustrate the basic concept of the potential function, consider a pool of rectangular shape with sides of length  $L_1$  and  $L_2$  along the  $x$  and  $y$  axes respectively and the origin as the lower left corner of the pool. The function

$$\beta(x, y) = x(x - L_1)y(y - L_2) \quad (2.4)$$

equals zero at any point on the pool border and therefore satisfies the first property.

However, this function is clearly discontinuous at the corners of the pool and in order to meet the requirements of the second property  $\beta(x, y)$  is combined with an auxiliary function of the form

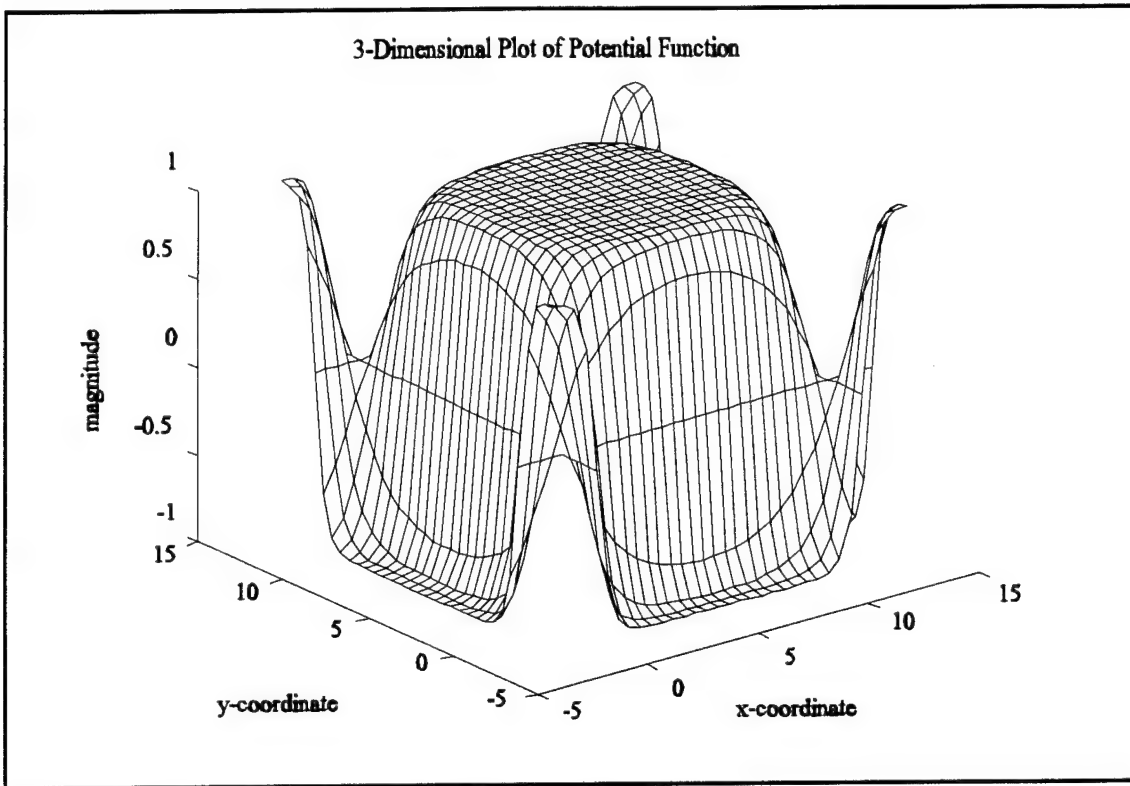
$$F_\lambda(z) = \frac{1 - e^{\frac{-\lambda}{z}}}{1 + e^{\frac{-\lambda}{z}}} \quad \lambda > 0 \quad (2.5)$$

Therefore,

$$V(x, y) = F_\lambda(\beta(x, y)) \quad (2.6)$$

which meets the requirements of all three properties, returning a value in the range  $[-1, 1]$

and is differentiable at any point within the environment. A typical potential function for a rectangular pool is presented as Figure 2.1.



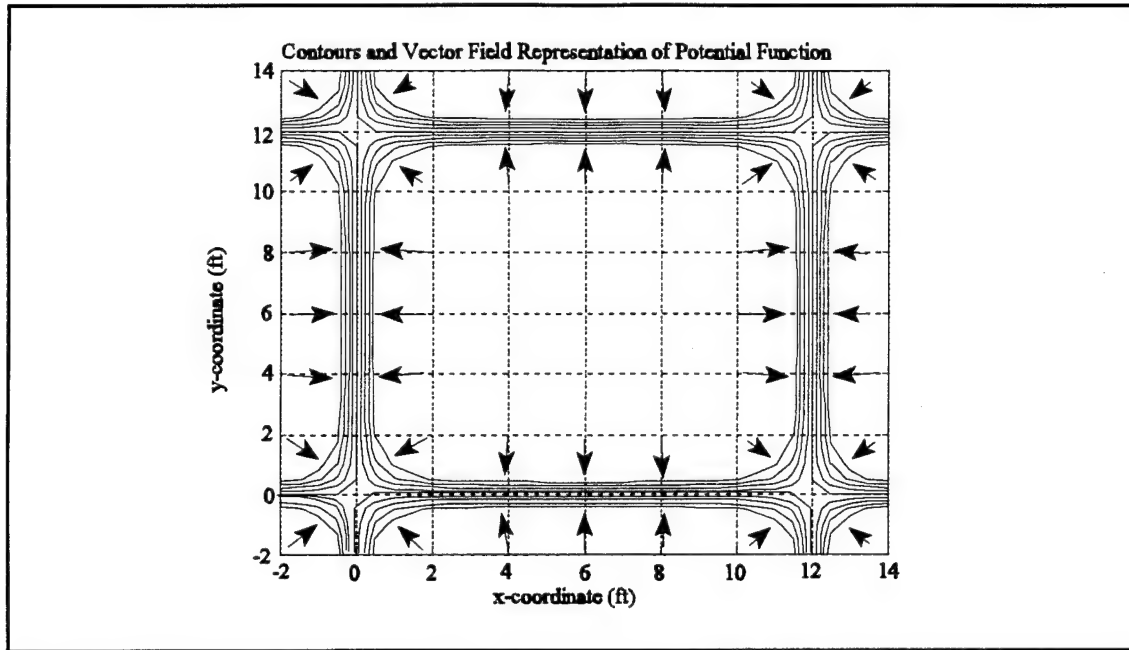
**Figure 2.1 Potential Function for Rectangular Pool**

Application of the spatial gradient operator to the potential function produces the attractive field of Figure 2.2 associated with this function. The attractive field magnitude will form the basis of the navigation correction factor which is the core of the algorithm discussed in detail in the following sections and implemented in scenarios of increasing complexity.

### **C. INTERACTION BETWEEN MODELS**

The sonar returns produced by the AUV installed sonar system provide the required interaction between the AUV dynamic model and the potential function model of the

environment. A summary of the operating characteristics of the high frequency scanning sonar installed on the NPS *Phoenix* AUV is included as Appendix B.



**Figure 2.2 Contours and Vector Field of Potential Function**

To illustrate the approach taken, consider a vector which defines the position of the vehicle at any given time

$$\mathbf{p}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \quad (2.7)$$

and define a measurement vector consisting of the sonar information

$$\mathbf{s}(t) = \begin{bmatrix} \rho(t) \\ \alpha(t) \end{bmatrix} \quad (2.8)$$

where  $\rho$  represents the range and  $\alpha$  the angle of the sonar return relative to the x-axis. The vector sum  $\mathbf{p} + T\mathbf{s}$  defines the position of the tip of the sonar vector in cartesian

coordinates with the matrix  $T$  representing the transformation between vehicle and inertial coordinate frames.

Now define a second measurement vector consisting of the data provided by the sensors monitoring velocity and heading of the vehicle

$$\mathbf{z}(t) = \begin{bmatrix} v(t) \\ \theta(t) \end{bmatrix} \quad (2.9)$$

This results in a model which represents the dynamics and kinematics of the vehicle operating in the defined environment as a system of equations in the form

$$\left\{ \begin{array}{l} \dot{\mathbf{p}} = f_1(\mathbf{z}) \\ \dot{\mathbf{z}} = f_2(\mathbf{z}, \mathbf{u}) \\ \mathbf{z}_m = \mathbf{z} + \mathbf{w}_z \\ 0 = V(\mathbf{p} + T(\mathbf{z})\mathbf{s}) \end{array} \right. \quad (2.10)$$

where the last equation is obtained by recalling that the value of the potential function is zero if the tip of the sonar vector falls on the boundary (sonar reflecting surface) of the environment.

#### D. CONVERGENCE

If we define the vehicle's estimated position as  $\hat{\mathbf{p}}$  then it can be updated as

$$\dot{\hat{\mathbf{p}}} = f_1(\mathbf{z}_m) + \mu \nabla V(\hat{\mathbf{p}} + T\mathbf{s}) \quad (2.11)$$

where  $\mu$  is a positive constant defined as the navigation correction factor gain (or *step size*). Clearly, if  $\hat{\mathbf{p}} = \mathbf{p}$  (no error in position estimation), the second term above will be zero and  $\dot{\hat{\mathbf{p}}} = f_1(\mathbf{z}_m)$  as expected (neglecting noise).

For the situation where an acceleration measurement is available, as would be the case with an inertial navigation system which includes accelerometers, the correction can be applied to the position and velocity estimate and can be updated as

$$\begin{cases} \dot{\hat{\mathbf{p}}} = \hat{\mathbf{v}} + \mu_1 \nabla V(\hat{\mathbf{p}} + T\mathbf{s}) \\ \dot{\hat{\mathbf{v}}} = \mathbf{a} + \mu_2 \nabla V(\hat{\mathbf{p}} + T\mathbf{s}) \end{cases} \quad (2.12)$$

with  $\mu_1, \mu_2$  being positive constants.

To demonstrate local convergence of this method (for the general case where a velocity measurement is available) define the position error as  $\tilde{\mathbf{p}} = \hat{\mathbf{p}} - \mathbf{p}$ . Combining equations (2.10) and (2.11) yields the error equation

$$\dot{\tilde{\mathbf{p}}} = \mu \nabla V(\hat{\mathbf{p}} + T\mathbf{s}) \quad (2.13)$$

and pre-multiplying by the transpose of the error yields

$$\tilde{\mathbf{p}}^T \dot{\tilde{\mathbf{p}}} = \mu \tilde{\mathbf{p}}^T \nabla V(\mathbf{p} + T\mathbf{s} + \tilde{\mathbf{p}}) \quad (2.14)$$

where we recognize that the point  $\mathbf{p} + T\mathbf{s}$  is on the reflective surface and that for  $\tilde{\mathbf{p}}$  sufficiently small that the right hand side of the above equation is non-positive by the third property specified for the potential function and  $|\tilde{\mathbf{p}}|^2$  is decreasing with time. Integration of equation (2.14) with respect to time yields

$$\begin{aligned} \frac{1}{2} (|\tilde{\mathbf{p}}(t)|^2 - |\tilde{\mathbf{p}}(0)|^2) = \\ \mu \int_0^t \tilde{\mathbf{p}}(\tau)^T \nabla V(\mathbf{p} + T\mathbf{s} + \tilde{\mathbf{p}}(\tau)) d\tau \end{aligned} \quad (2.15)$$

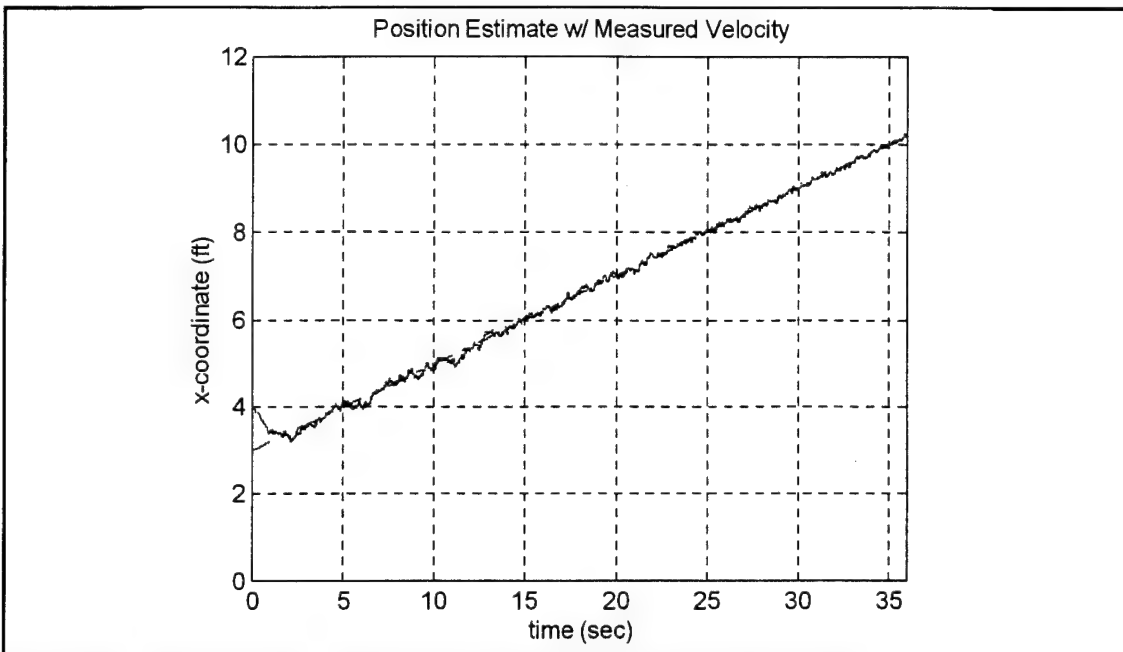
and since the integrand is nonpositive for all  $\tau$  it will go to zero as time tends to infinity. This implies that the error vector  $\tilde{\mathbf{p}}$  tends to be orthogonal to the gradient of the field, and that the point  $\hat{\mathbf{p}} + T\mathbf{s}$  tends to be on the reflecting surface. The mathematical verification for the proof of convergence in the special case of acceleration vice velocity measurements is addressed in [Ref. 2]. For the majority of simulations addressed in this thesis, velocity measurements will be assumed available and utilized for position estimation. This

assumption is critical to the accurate navigation algorithm performance in data-sparse environments to be addressed in Chapter IV.

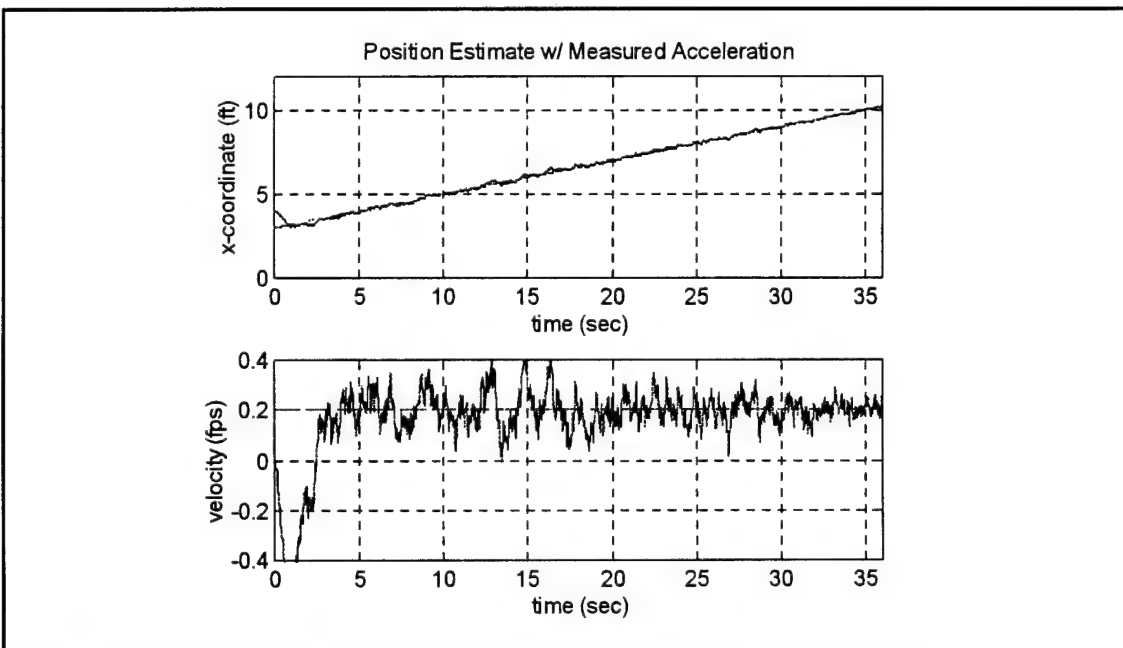
As a graphical demonstration of convergence, consider the following simple scenario. An AUV is moving in a horizontal path with constant velocity towards a reflective boundary (wall). The initial conditions are:

$$\left\{ \begin{array}{l} x(0) = 3 \text{ ft} \\ v(0) = 0.2 \text{ ft/s} \\ \hat{x}(0) = 4 \text{ ft} \end{array} \right. \quad (2.16)$$

Application of the position estimation algorithm (velocity measured) with an integration step size of 0.0225 seconds (also the sonar ping interval) yields the estimated position trajectory of Figure 2.3. The position estimate converges rapidly to the actual position. Repeating the simulation of the same scenario but with acceleration vice velocity measurements results in the trajectories for estimated position and velocity of Figure 2.4. Note that the position estimate converges in approximately the same amount of time as for the previous case, while the velocity takes slightly longer to converge and exhibits greater statistical variation as can be anticipated from theory.



**Figure 2.3 Verification of Convergence with Measured Velocity**



**Figure 2.4 Verification of Convergence with Measured Acceleration**

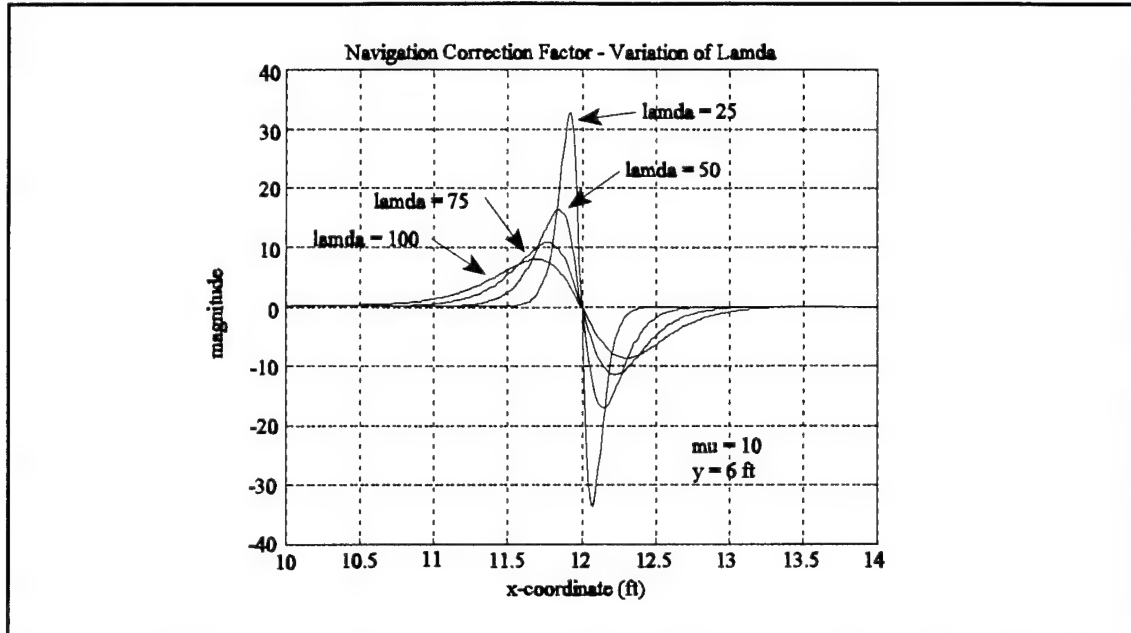
## E. IMPORTANCE OF PARAMETER SELECTION

The proper selection of the parameter  $\lambda$  used in Equation (2.5) to describe the auxiliary function is of vital importance in that it affects the width of the attractive region around the boundary or reflective surface of the potential function approximating the environment. If the tip of the sonar vector does not fall within the attractive region then negligible correction to the estimated position of the vehicle will occur. However, if the attractive region is too wide then it may include objects or obstacles which are not modeled in the environment. A sonar return off such an obstacle could be construed as a return off the reflective surface generating the field, thereby producing an erroneous correction factor.

In previous navigation algorithms designed for implementation on the NPS AUV [Refs. 3 and 4] the non-linear potential function of sonar information was utilized as the basis for correction factors in an Extended Kalman Filter (EKF) where the magnitude of  $V(\mathbf{x})$  had a lesser effect on algorithm performance than for our application. For the current design the magnitude as well as the width of the attractive field is crucial to the choice of a constant gain correction factor defining an appropriate error correction over a wide range of position errors. In addition, as the complexity of the environment increases the values of  $\lambda$  and  $\mu$  in Equations (2.5) and (2.11) must be carefully selected due to the multiplicative effect of potential functions based on multiple objects, as will be discussed in Chapter IV.

Figure 2.5 shows the effect of varying  $\lambda$  on the magnitude and width of the attractive region formed by the rectangular environment potential function at the reflective boundary. The factor  $\mu$  represents the size of the correction (usually called the *step size*) and it affects both the convergence rate and the stability of the algorithm.





**Figure 2.5 Effect of Varying Lamda on the Navigation Correction Factor**

In the next chapter these theoretical concepts will be implemented in an algorithm which utilizes the potential function approach to solve the two-dimensional short range navigation problem of an AUV.



### **III. APPLICATION OF POTENTIAL FUNCTIONS**

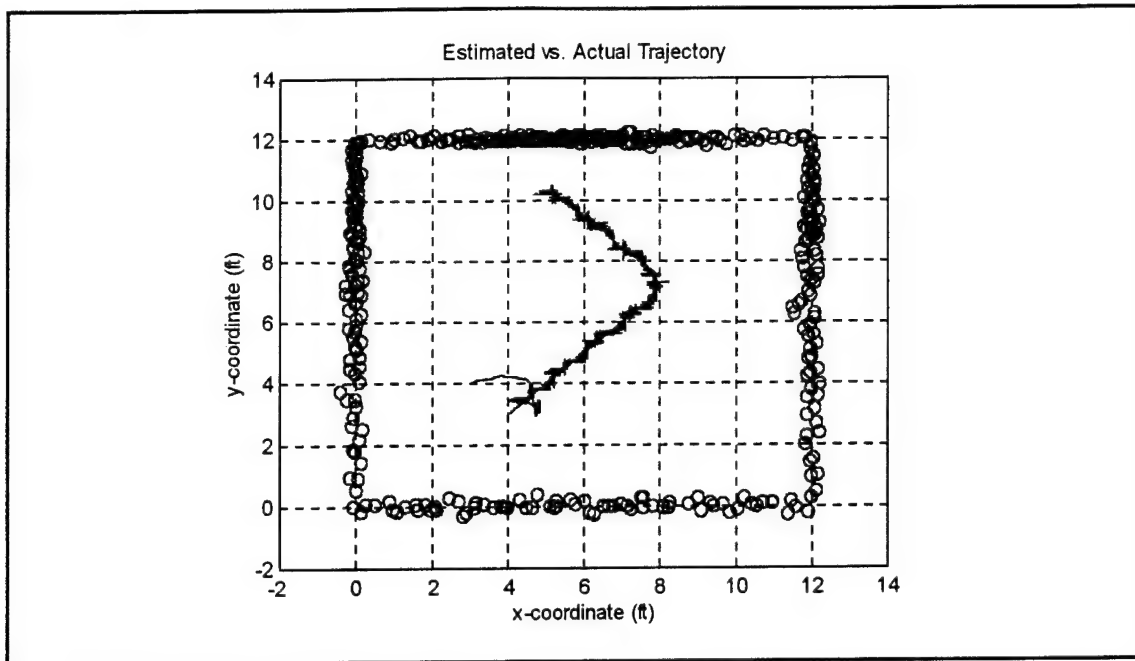
#### **A. ENVIRONMENTS WITH BORDERS**

In this chapter we address the problem of applying the potential function to a structured environment. Although we will be using the model of the test tank at the Naval Postgraduate School Annex, the results can be generalized to more complex environments, such as pipelines, oil rigs, harbors, etc. The rectangular environment with closed borders is the simplest realistic operating area which can be modeled with the potential function approach and would reflect operations such as acoustic test data-gathering with the NPS *Phoenix* AUV in the laboratory pool.

#### **B. SIMULATION RESULTS**

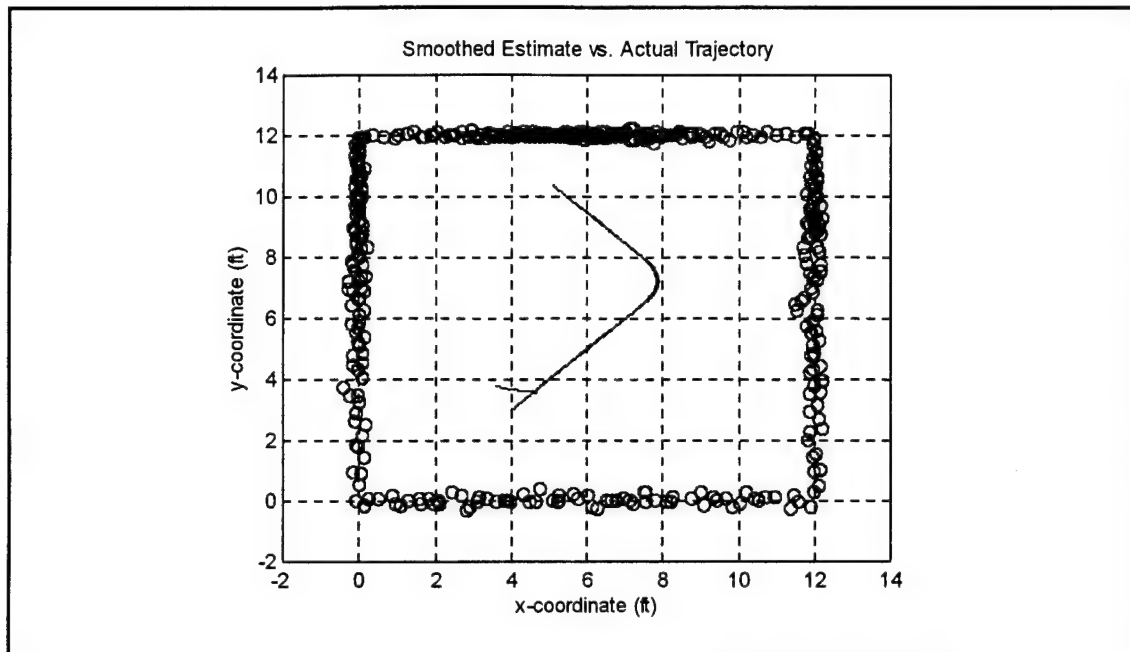
The navigation algorithm is tested on a rectangular operating environment of size 12 feet by 12 feet. The motion of the AUV is simulated with a non-linear state-space model with sensor noise applied to the velocity and heading measurements. The sonar system operation is also simulated with noise applied to both range and bearing measurements. All sensor noise is modeled with a gaussian zero-mean distribution. The SIMULINK models and MATLAB code utilized for the simulation are included as Appendix C. Figure 3.1 shows the results of a 72 second simulation which encompasses eight 360-degree sonar scans of the environment border and graphically compares the actual and estimated trajectories of the vehicle as well as plotting the estimated borders of the pool.

As can be seen from the plot, the algorithm quickly drives the estimated trajectory to the actual track of the vehicle. Of interest is the "staircase-like" appearance of the estimated trajectory. This is a result of the fact that when sonar returns are falling on the vertical borders of the pool only corrections to the x-dimension of velocity, and hence position, can be obtained. Similarly, sonar returns off of the horizontal boundaries correct only the y-dimension components of position and velocity. This effect is minimized by



**Figure 3.1 Estimated vs. Actual Vehicle Trajectory**

utilizing a smoothing filter to process the estimated position results. The effects of applying the algorithm are presented in Figure 3.2 which compares the actual track to the smoothed estimate for the same scenario utilized for the simulation in Figure 3.1.



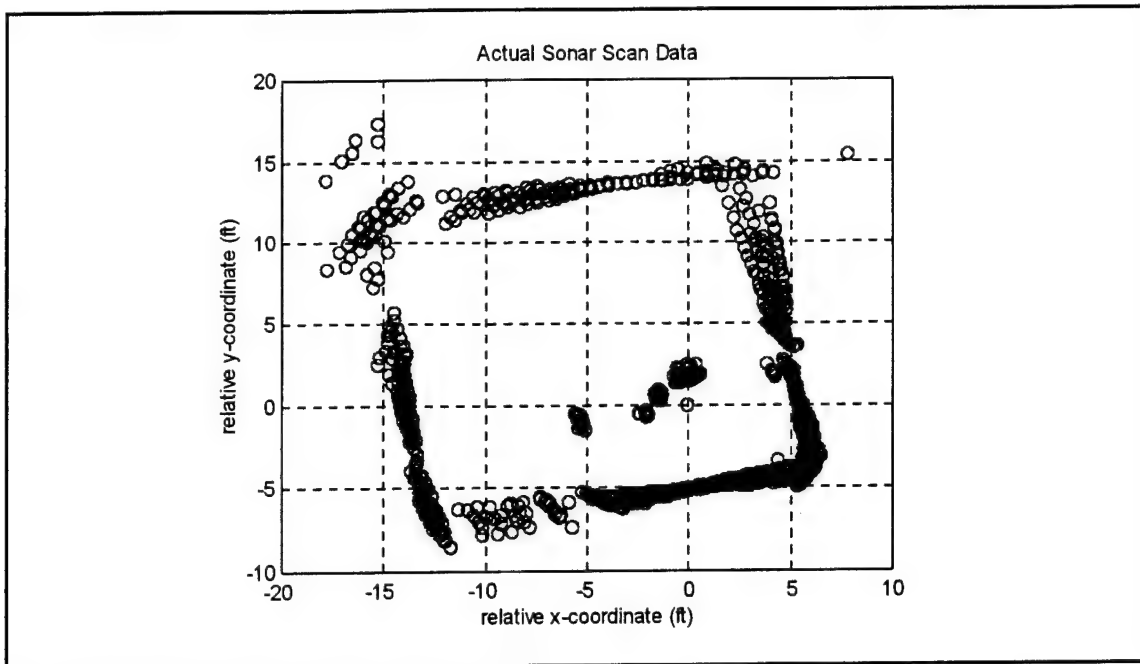
**Figure 3.2 Effect of Smoothing Position Estimates**

Smoothing of the estimated position data results in an estimate that follows the actual trajectory with a high degree of accuracy. The disadvantage of data smoothing is that it generates a time delay in producing a position estimate, since the filter utilizes the previous 400 data points (one sonar scan worth of data) for weighted averages. The details of the filter design are included as Appendix D.

### **C. SIMULATIONS WITH ACTUAL DATA**

In order to demonstrate the robustness of the navigation algorithm, simulations with actual sonar data gathered in the test pool were conducted. In addition, velocity and heading sensor information is not used in forming the position estimate, which is a more stringent requirement than the standard initial simulation assumptions of available data. Figure 3.3 shows a typical data run for the AUV operating in a six meter by six meter test pool.

The available sonar data is excessively corrupted by noise as a result of reverberation caused by confining the sonar acoustic energy in a relatively small water volume. This effect is evident through the multiple false returns both inside and outside of the pool boundaries.

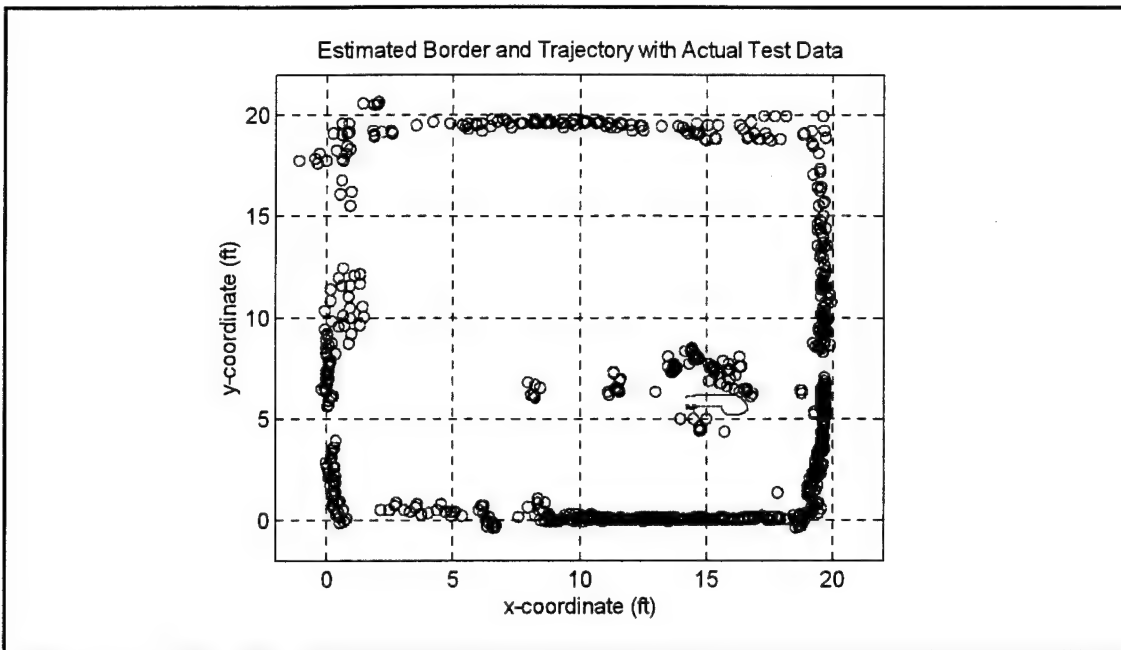


**Figure 3.3 Actual Sonar Test Data**

Interpretation of the sonar data is further complicated by an obstacle placed in the lower right-hand quadrant which is not modeled in the potential function.

When vehicle velocity and heading information are available, the navigation correction factor adjusts the sensed velocity cartesian components to drive the distance between the estimated position and environment boundary to match the sonar range. Without velocity information a higher navigation correction factor gain is required to compensate for the lack of a velocity signal driving the estimated position of the vehicle. Application of the navigation algorithm to the sonar data of Figure 3.3 results in the estimated environment and vehicle trajectory of Figure 3.4.

Although there are still a number of false sonar returns and gaps in the estimated border of the environment, the algorithm is clearly able to ascertain the track of the vehicle and the correct size and orientation of the test pool without utilizing velocity and heading data input.



**Figure 3.4 Simulation with Actual Sonar Test Data**

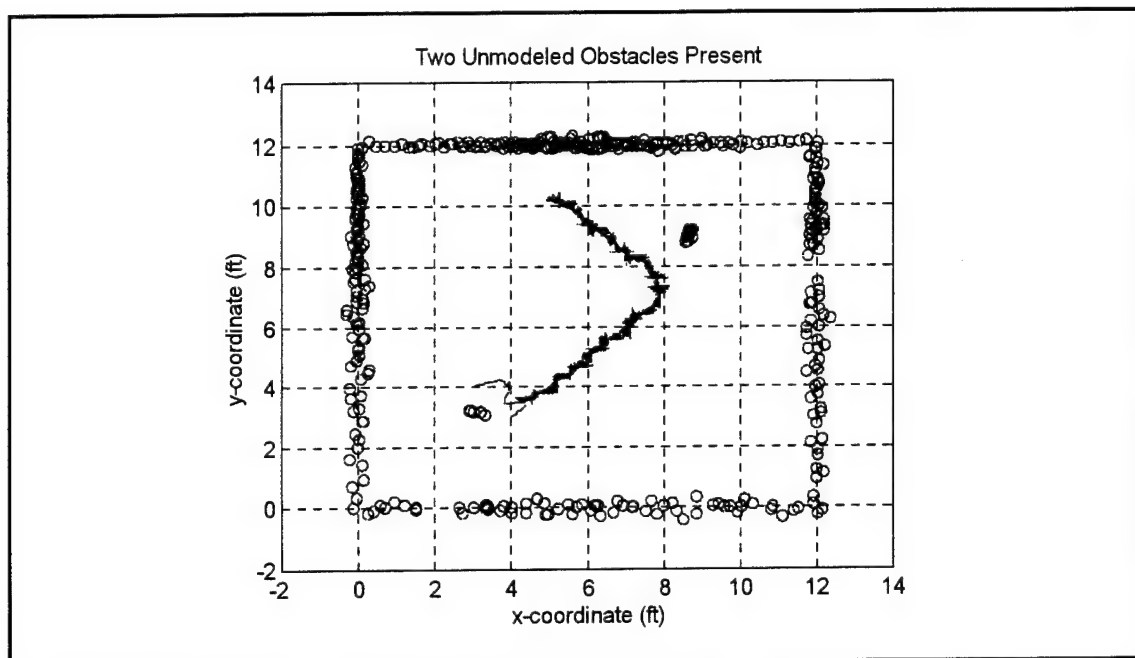
#### **D. INCORPORATION OF OBSTACLES IN THE OPERATING ENVIRONMENT**

An obstacle is in general an object in the environment which will reflect the sonar's acoustic energy. Obstacles will fall into two categories: 1) Obstacles which are not modeled in the potential function description of the environment, and 2) Obstacles which are accounted for in the environment description. The effects of both types will be addressed in the subsequent sections.

##### **1. Unmodeled Obstacles**

Obstacles which fall into this category include objects in the vehicles operating environment which were not considered when defining the components of the environment which will provide navigation correction factors to the estimated position of the vehicle. Examples include underwater mines, sunken vessels, uncharted sea mounts, etc. The difficulty when considering such objects is that if they happen to be located close to the modeled environment borders or components, the sonar return off of such an object could result in an erroneous navigation correction factor since it falls in the attractive region of the

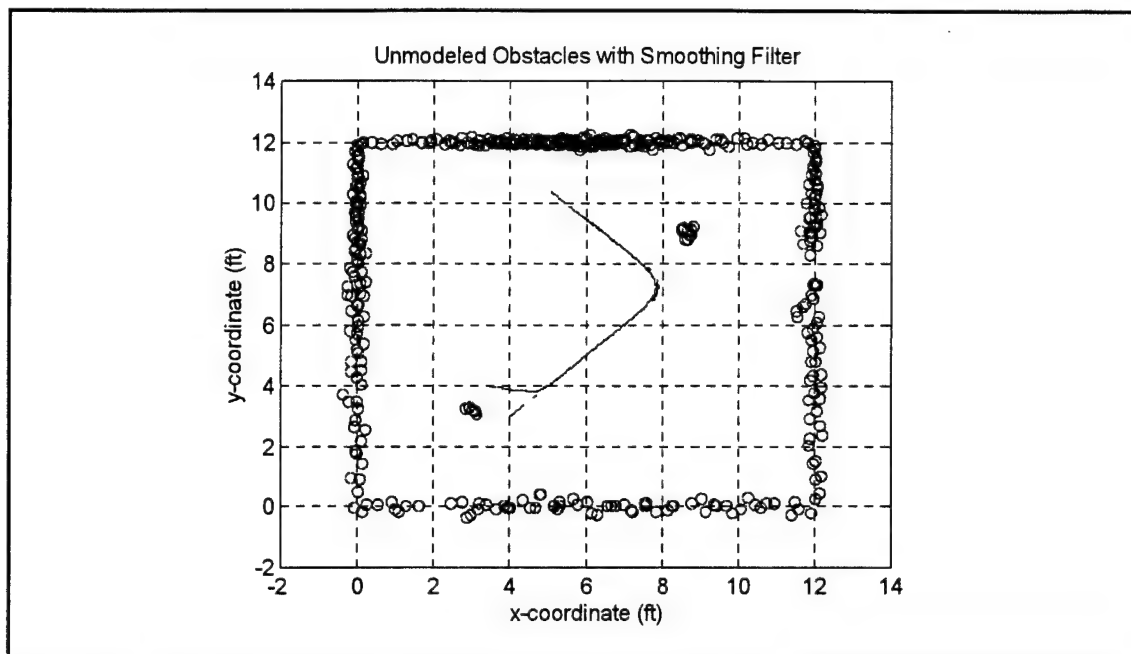
environment's potential function. For this reason it is essential that parameter  $\lambda$  in the potential function is chosen so that the attractive region is wide enough to provide adequate correction, but narrow enough to be robust in the presence of unmodeled obstacles. Figure 3.5 illustrates a pool simulation with unmodeled obstacles at coordinates (3,3) and (9,9). The effects of applying the smoothing filter to the estimated track of the vehicle are presented as Figure 3.6.



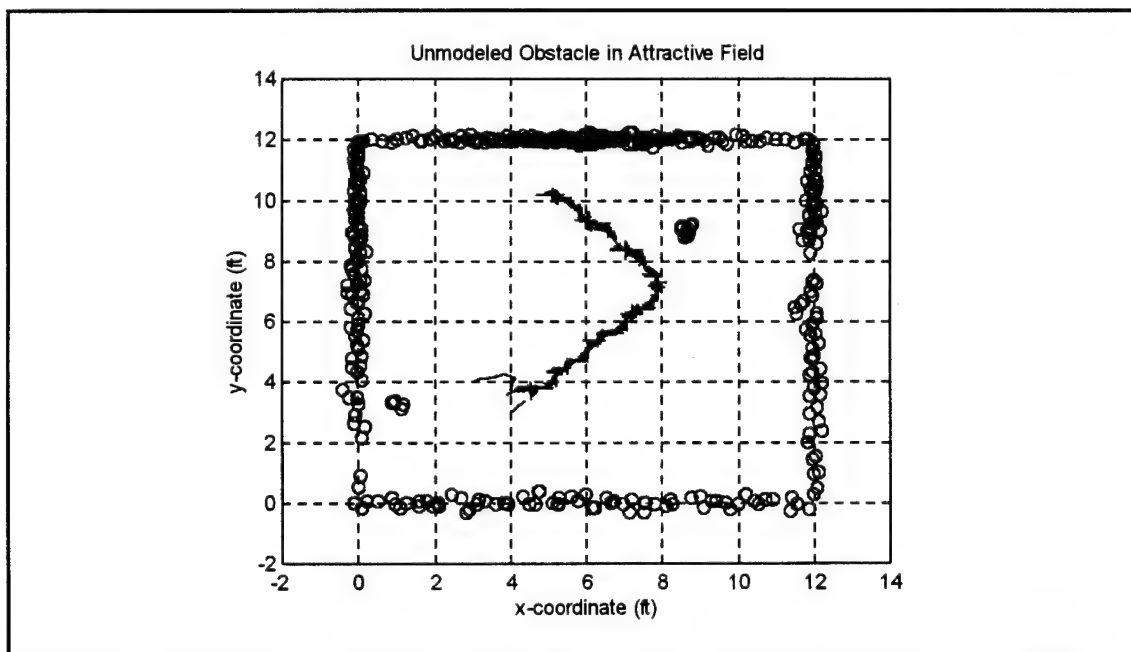
**Figure 3.5 Simulation with Unmodeled Obstacles**

In order to ascertain the effect of an obstacle which falls in the attractive region of the potential function, the simulation was re-run with the lower left-hand obstacle moved to coordinates (1,3) which is in the attractive field of the left pool edge. Figure 3.7 shows the results of this simulation. Note that the erroneous correction results in greater statistical fluctuation in the x-dimension yet the algorithm is still capable of producing convergence of the position estimate with the actual trajectory. The effect of applying the smoothing filter is seen in Figure 3.8 where virtually no difference is distinguishable from the less complex scenario of Figure 3.6.

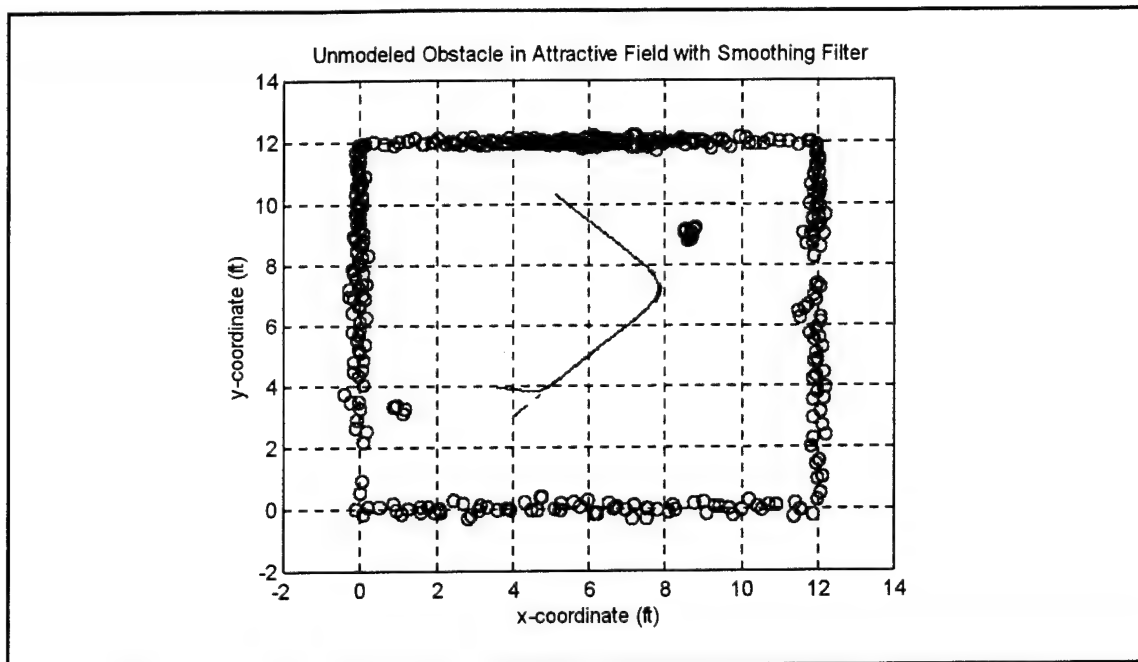




**Figure 3.6 Unmodeled Obstacles with Smoothing Applied**



**Figure 3.7 Simulation with Unmodeled Obstacle in Attractive Field**



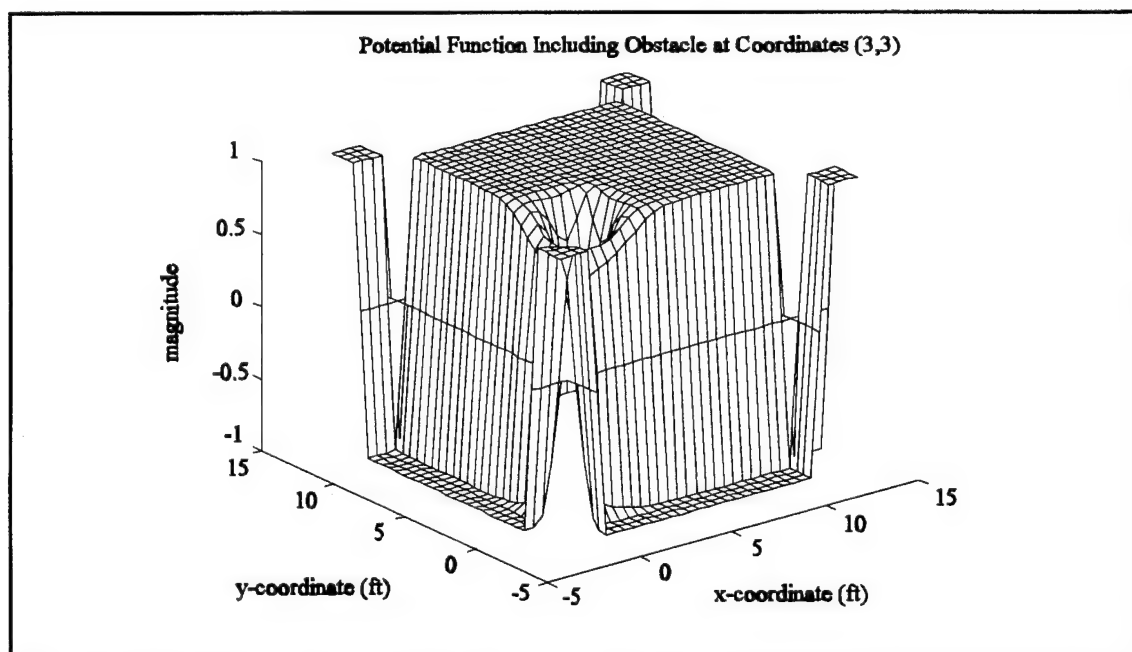
**Figure 3.8 Unmodeled Obstacle in Attractive Field with Smoothing Applied**

## **2. Obstacles Modeled in Potential Function**

Objects in the operating environment which are in known locations can be utilized to provide geographic references to the navigation algorithm in the same way that the pool boundaries provide a frame of reference for the test pool case. These objects can either be those already in the operating environment such as geographic features, pier pilings, quay walls, etc. or those intentionally place in the environment by the AUV operators such as marker bouys or sonar reflectors. This section is primarily concerned with the latter to demonstrate the ability to use an object place in the operating area at a known location to provide geographic reference to the navigation algorithm in a data-poor operating environment. As shown in Chapter II, incorporating additional geometric features in the potential function is relatively simple. The obstacles will be modeled as a six-inch diameter cylinder located in the horizontal operating plane of the vehicle. The details of the potential function calculation will be deferred to the next chapter where potential function descriptions for a number of geometric primitives will be addressed. Figure 3.9 demonstrates the effect of including an obstacle (marker) in the potential function. The attractive "well"

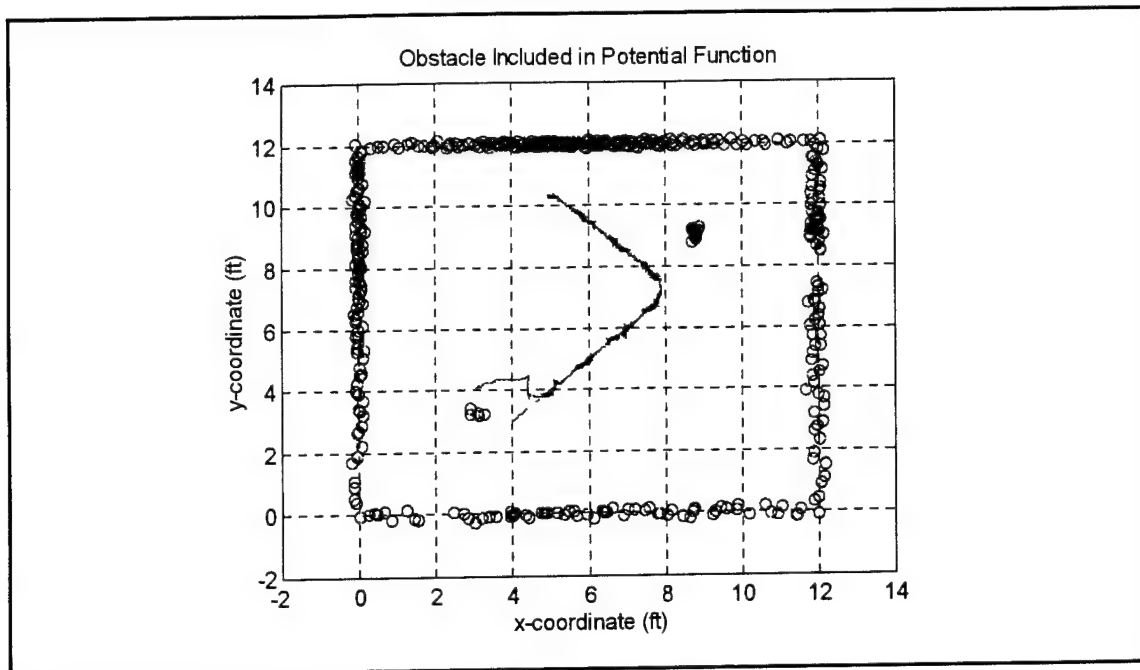
of the marker at coordinates (3,3) is clearly seen in the 3-dimensional plot of the potential function.

Figures 3.10 and 3.11 illustrate a simulation with markers (obstacles) at the same location as in the simulation of Figure 3.5 and the marker at coordinates (3,3) modeled in the potential function. The values of  $\lambda$  and  $\mu$  are chosen such the the marker provides the majority of the navigation correction. By comparing this to the aforementioned simulation it can be seen that the algorithm converges more slowly until the sonar beam falls on the marker, at which time the estimated position rapidly converges to the actual vehicle position. In addition, there is less statistical variation in the estimated trajectory since larger corrections occur only over a small portion of the sonar scan (when the obstacle falls within the acoustic beam of the sonar).

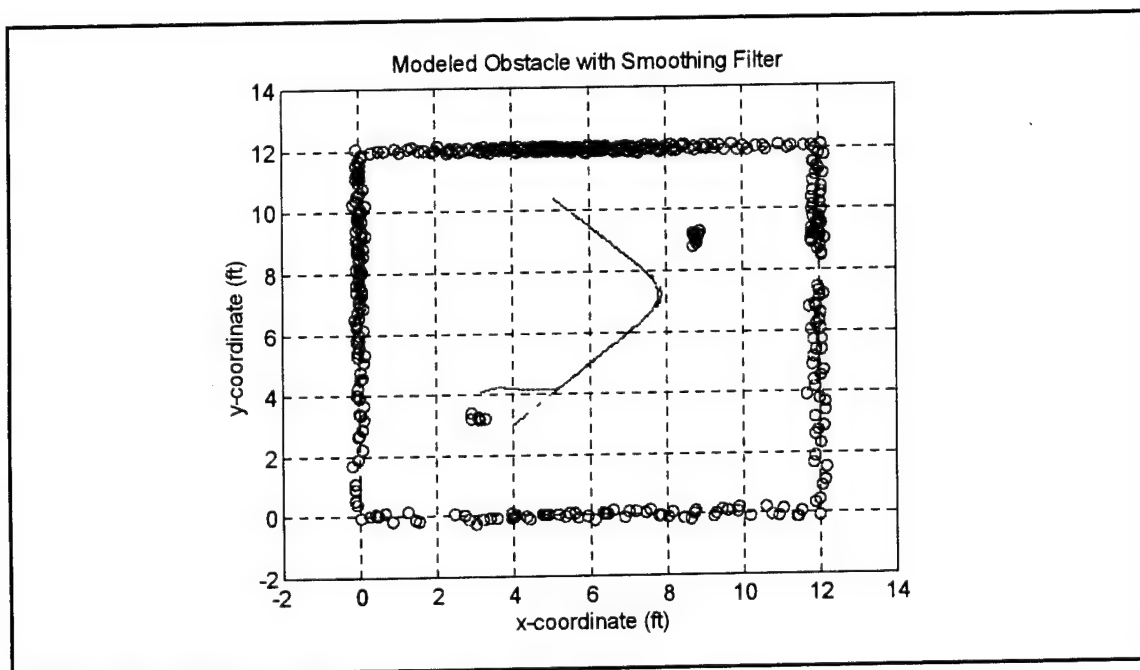


**Figure 3.9 Potential Function with Modeled Obstacle**

In the next chapter the concepts introduced here are extended to environments and scenarios of greater complexity in that data required for updating the vehicle estimated position will not be continuously available.



**Figure 3.10 Obstacle Modeled in Potential Function**



**Figure 3.11 Modeled Obstacle with Smoothing Applied**

## **IV. APPLICATIONS IN COMPLEX ENVIRONMENTS**

### **A. ENVIRONMENTS WITHOUT BORDERS**

In an environment without continuous borders the navigation correction factor is applied only when the sonar acoustic beam falls on an obstacle in the vehicle operating area. These obstacles can either be placed in the environment as a marker or pre-existing geographic features. The availability of heading and velocity sensor data is critical to the proper operation of the navigation algorithm in an open environment because corrections to the estimated position of the vehicle will occur only over a relatively small portion of each 360-degree sonar scan. In addition, currents and tidal effects in the operating area can generate additional position and velocity errors which must be considered.

### **B. SIMULATION RESULTS**

The algorithm is applied to an open environment which includes two marker buoys to provide the required navigation references. Each buoy is a six-inch diameter cylinder placed in the horizontal operating plane of the vehicle. Figure 4.1 shows the potential function for this scenario where the attractive wells of the two obstacles at coordinates (5,5) and (20,20) are clearly seen. Figure 4.2 illustrates the contours and associated attractive vector field of this function. The results of a 90 second simulation consisting of ten 360-degree sonar scans are presented as Figure 4.3. Application of the smoothing filter is unnecessary in this case since the staircase-like appearance of the estimated trajectory is not evident without continuous navigation corrections.

#### **1. Accounting for Effects of Currents**

Currents in the vehicle operating area require a modification to the state-space model utilized to generate the actual trajectory of the AUV. The current is assumed to be slowly varying with time and can be measured or estimated prior to actually operating the vehicle in the environment. Adding two components to the state vector to account for the current components in the x and y dimensions yields the modified state-space model:

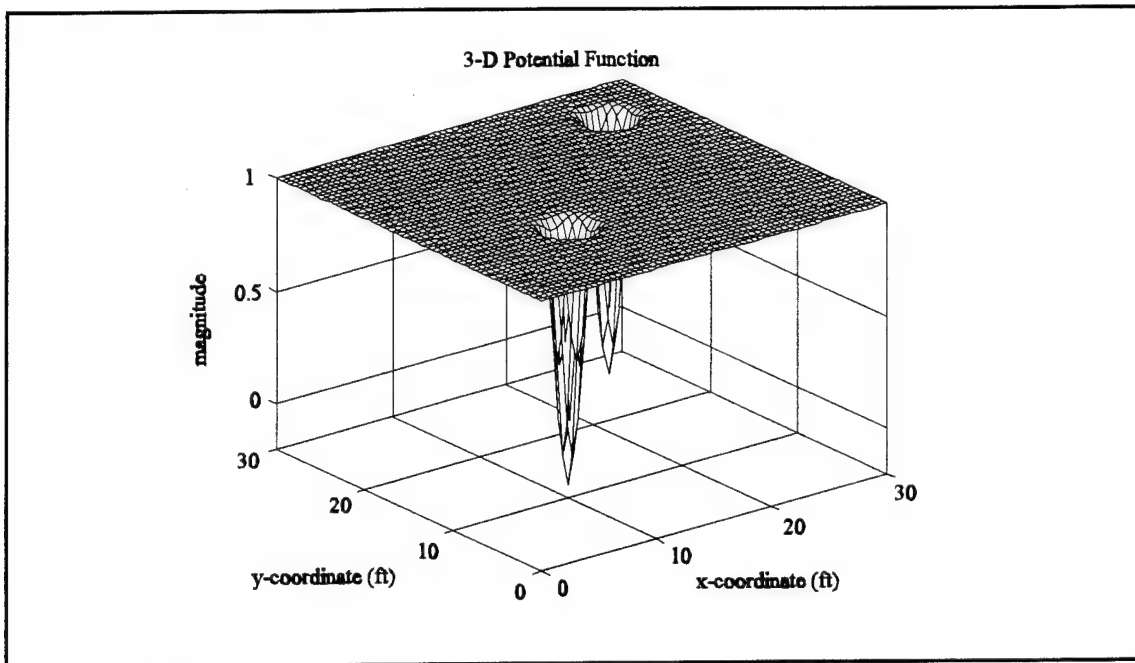


Figure 4.1 Open Environment Potential Function with Two Obstacles

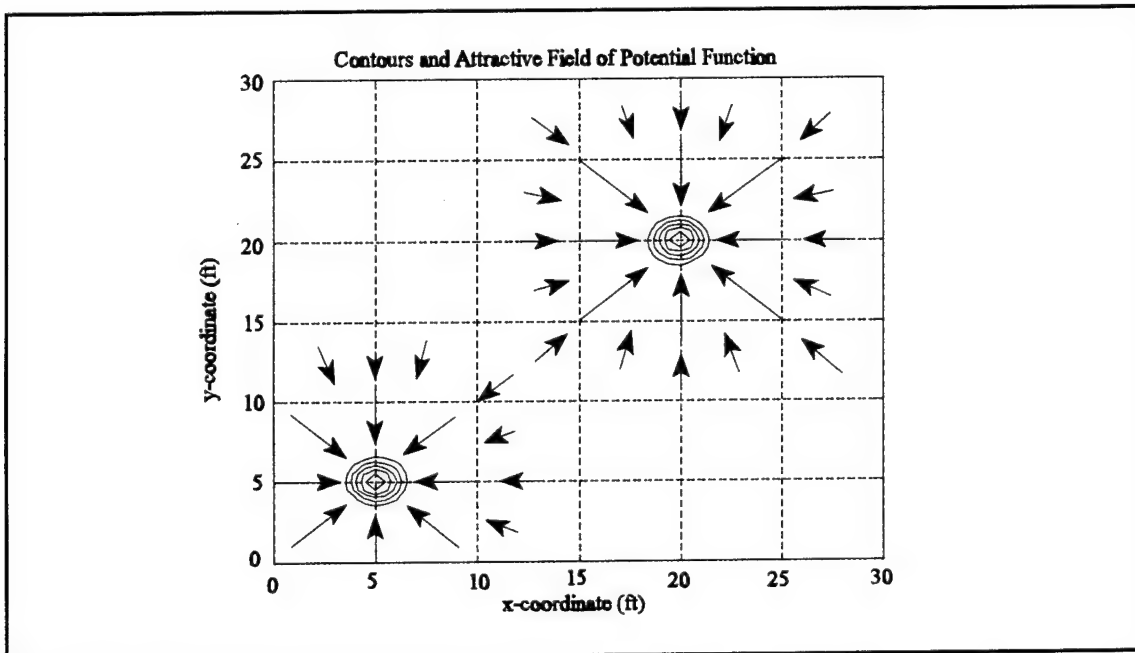
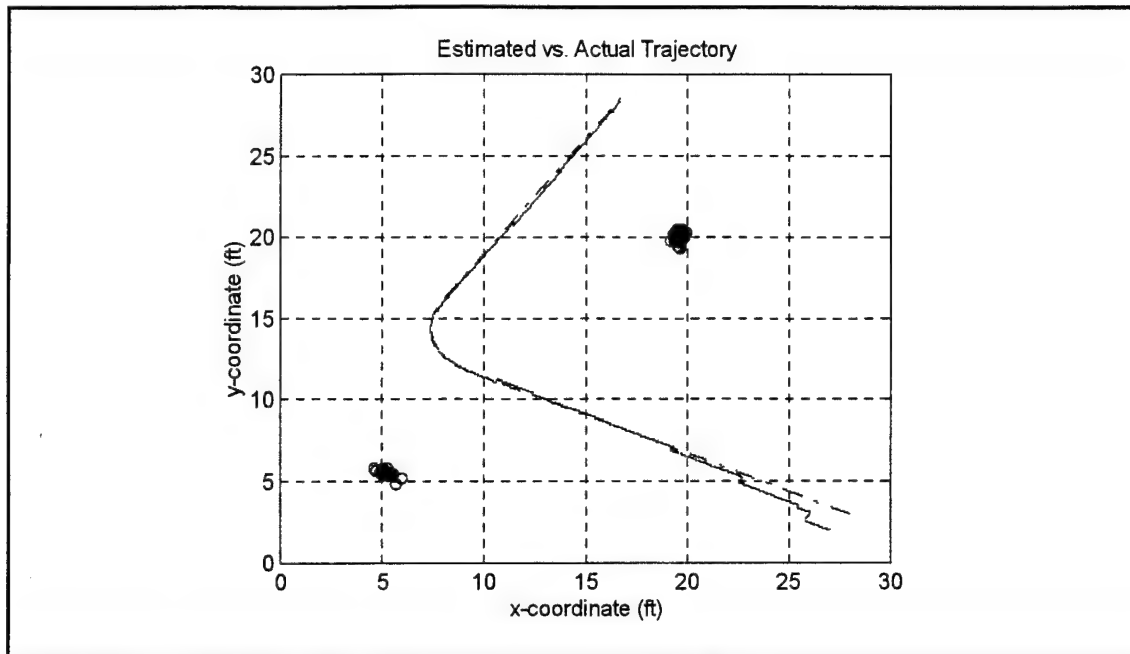


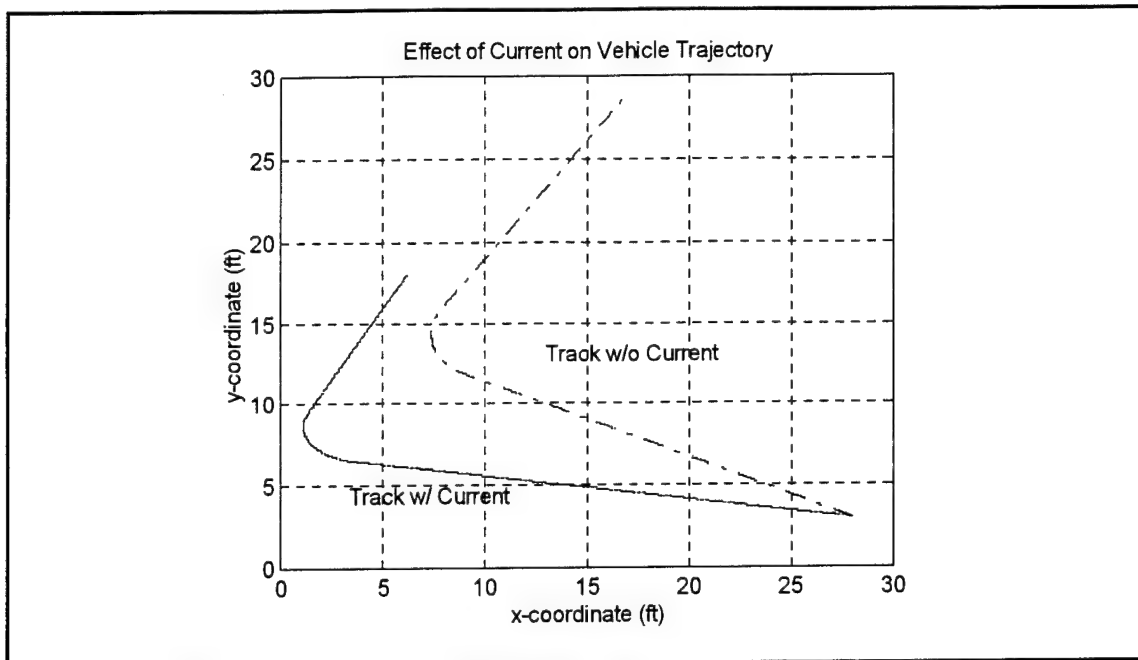
Figure 4.2 Contours and Attractive Field of Potential Function



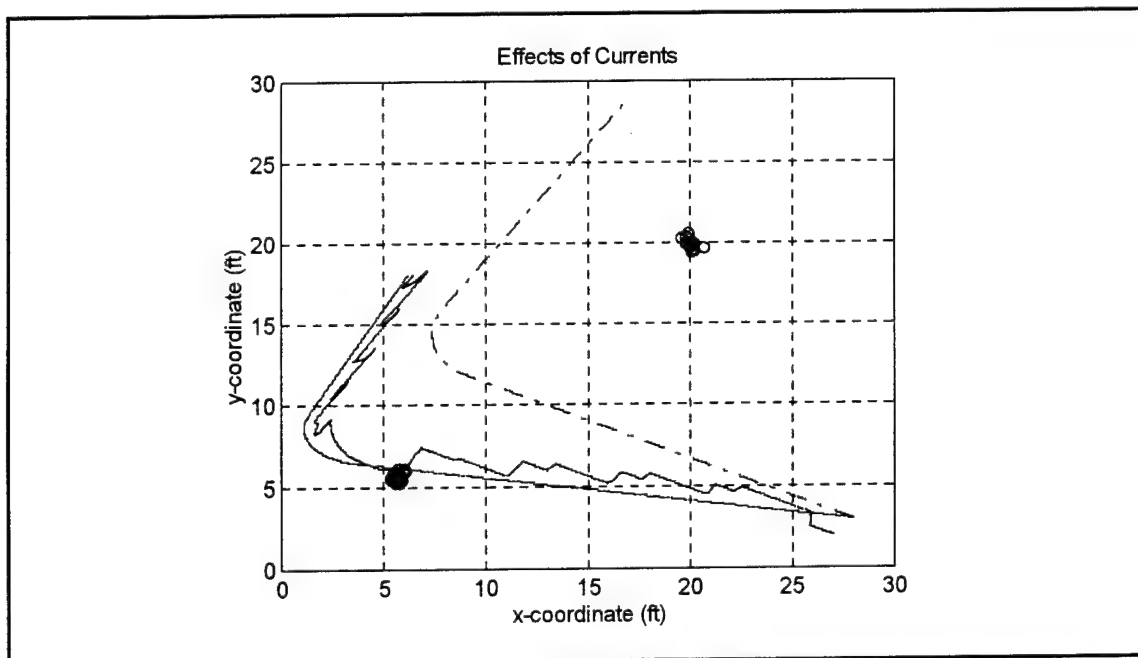
**Figure 4.3 Simulation in Environment without Borders**

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & \cos \theta_0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \sin \theta_0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -508.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1.667 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{u} + \mathbf{w} \quad (4.1)$$

Rerunning the simulation of Figure 4.3 and utilizing the revised vehicle model with a small current traversing the operating area results in the actual vehicle trajectory of Figure 4.4. The results of applying the navigation algorithm are presented as Figure 4.5 for this scenario.



**Figure 4.4 Effect of Current on Vehicle Track**



**Figure 4.5 Simulation with Current in Operating Area**



As we can see in these simulations, the current has a significant effect on the estimated trajectory during the portions of the sonar scan where corrections are not available. This is due to the difference between the vehicle sensor velocity as measured relative to the water and the inertial velocity measured relative to the ground which includes the effect of the current. Each pass of the sonar beam over an obstacle causes the estimated position to "jump" towards the actual trajectory. Increasing the number of markers or navigational references counters this effect by providing more frequent correction to the position estimate.

### C. COMPLEX ENVIRONMENTS

The rendering of a realistic environment into an effective mathematical function which accurately describes its critical geographic features requires additional complexity over the potential functions investigated previously.

Recall that the describing function of the environment must equal zero if the tip of the sonar vector falls on the acoustic reflecting border of the environment, i.e.

$$\beta(x, y) = 0 \quad (4.2)$$

Consider an environment consisting of  $n$  components, each with their own describing mathematical function and the overall function being the product of each component individual function:

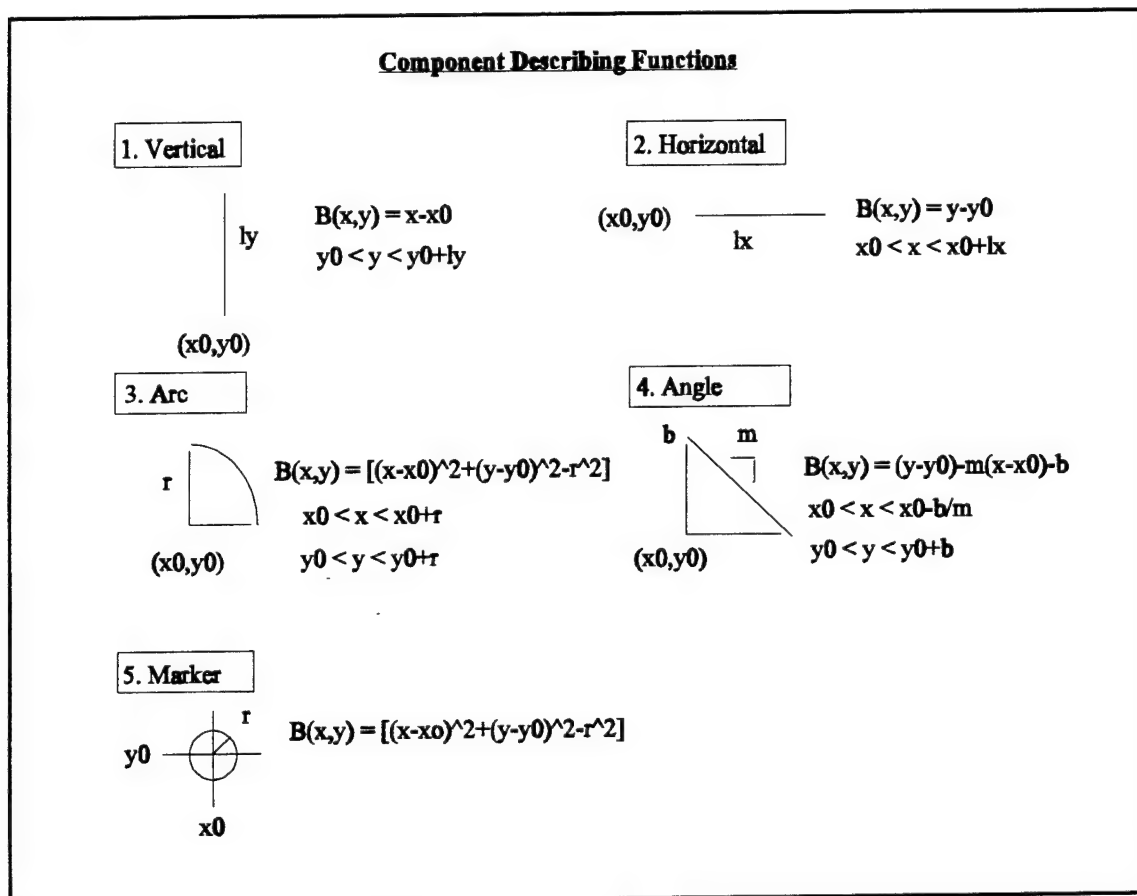
$$\beta(x, y) = \prod_{i=1}^n \beta_i(x, y) \quad (4.3)$$

Utilizing this approach results in a function which will be zero if the tip of the sonar vector falls on any component of the overall environment model. Application of the auxiliary function of Equation (2.5) results in a potential function meeting the three requirements set forth in Chapter II for the environment describing function.

Therefore,

$$V(x, y) = F_{\lambda} \left( \prod_{i=1}^n \beta_i(x, y) \right) \quad (4.4)$$

The five model components, or geometric primitives, chosen to mathematically describe the environment are summarized in Figure 4.6.

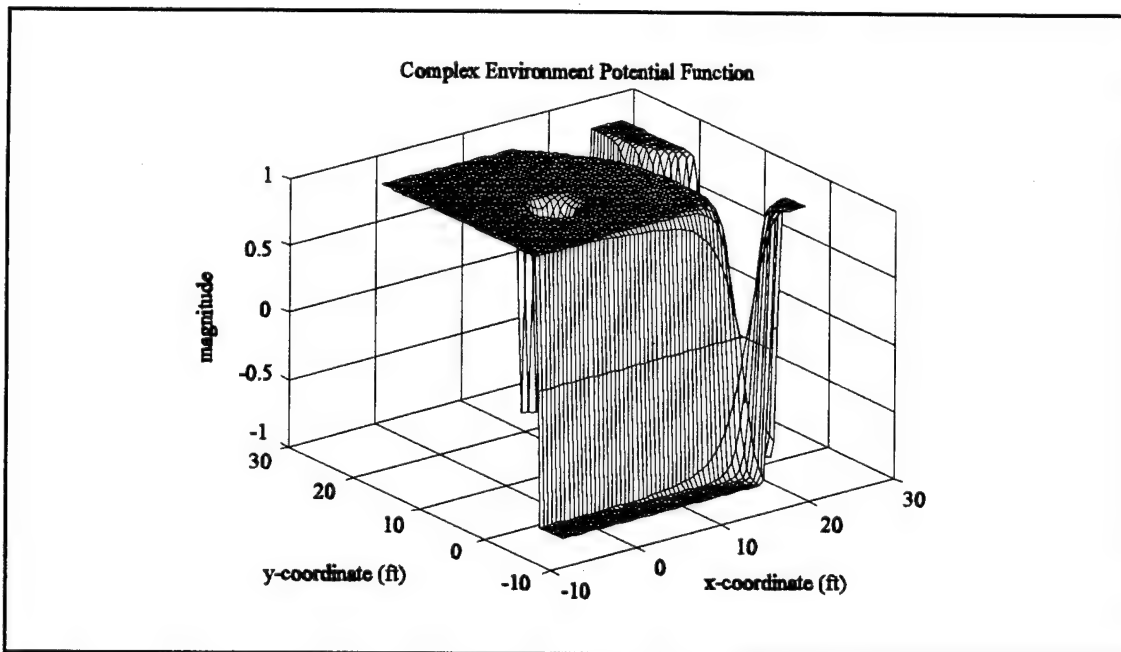


**Figure 4.6 Environment Model Components**

A wide variety of operating environments and geographic features can be modeled utilizing these components. For example a harbor scenario could be modeled which incorporates piers, quay walls, moored vessels, fixed navigation aids, bottom contours, etc.

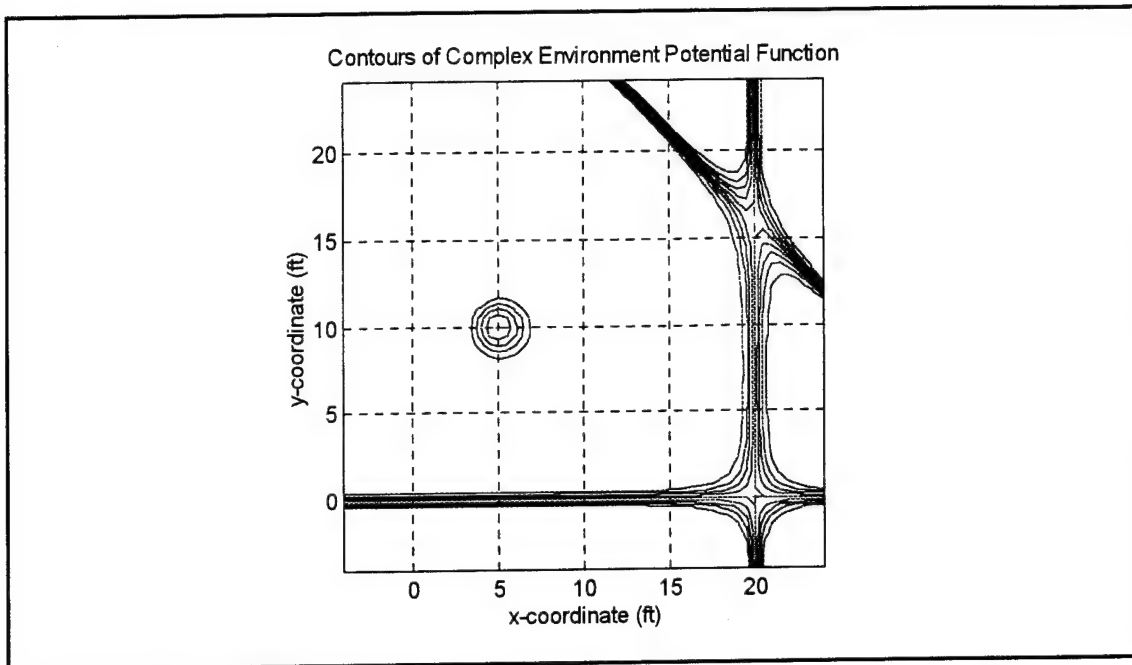
#### D. COMPLEX SIMULATION RESULTS

A scenario similar to the one discussed in the previous section was simulated with the AUV operating in a modeled harbor. The environment consists of a vertical segment, a horizontal segment, an angled segment and one marker buoy. This operating area is open on two sides. Currents are considered to be negligible as might be the case in a sheltered harbor. Figures 4.7 and 4.8 show the potential function and its associated contours for this scenario. The results of applying the navigation algorithm are presented as Figure 4.9.

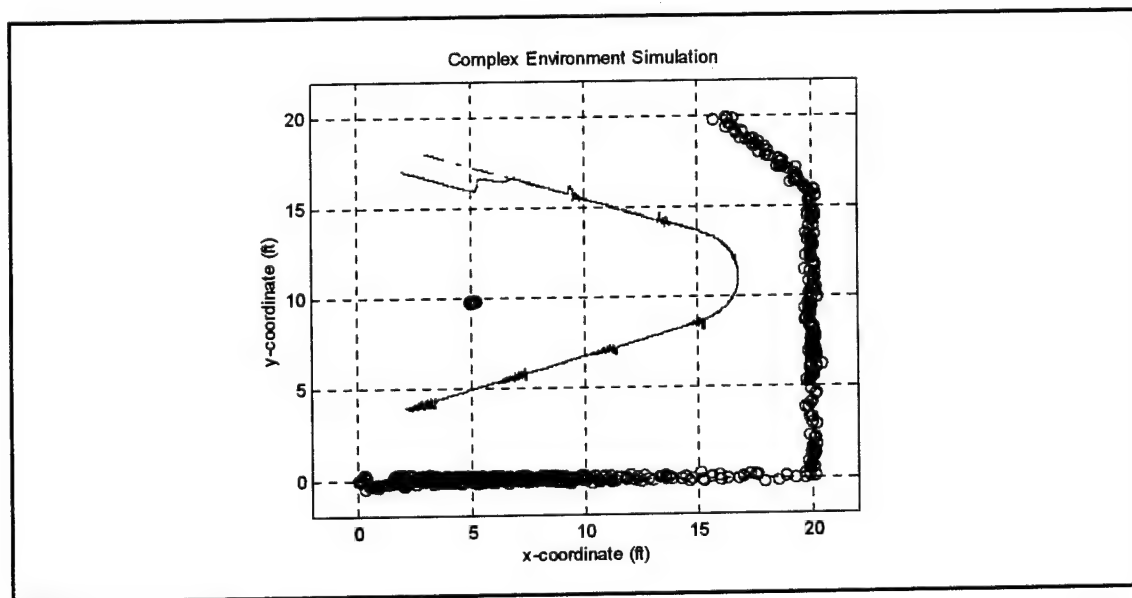


**Figure 4.7 Complex Environment Potential Function**

Because the sonar pointing angle is initialized at zero degrees, approximately one-half of a sonar scan passes before any correction to the estimated position occurs. As soon as the sonar returns off modeled environment components are obtained, the estimated trajectory rapidly converges to the actual vehicle track.



**Figure 4.8 Contours of Potential Function**



**Figure 4.9 Complex Environment Simulation**

In this chapter we have extended the algorithm designed with the potential function approach to apply to a number of general scenarios of increasing complexity. The robustness of this approach was verified through computer simulation of these scenarios. In the next chapter an alternative estimation strategy will also be investigated.



## V. APPLICATION OF SLIDING MODE THEORY

### A. SLIDING OBSERVERS FOR NONLINEAR SYSTEMS

As previously mentioned, the estimator developed in this research is not based on the Kalman Filter, unlike most available approaches. Simplicity of implementation, proven robustness and better numerical performance are the basis of this choice. In this chapter we present a nonlinear estimator (observer) based on sliding mode theory, widely applied in robust control systems, but still not widely used in state estimation. Initial research in this area shows that sliding observers have promising properties in the presence of modeling errors and sensor noise [Ref. 5]. The one-dimensional case will be considered here and then extended to two-dimensions with a closed-border simulation presented as an example application of the sliding mode observer.

#### 1. One-Dimensional Example

Consider a simple case of a vehicle moving along the x-axis towards a reflective surface from which sonar returns are obtained. This scenario is presented graphically in Figure 5.1. In lieu of the potential function previously described, the boundary is described mathematically by the function

$$V(x) = |L_1 - x| \quad (5.1)$$

where  $L_1$  is the x-coordinate of the reflective surface.

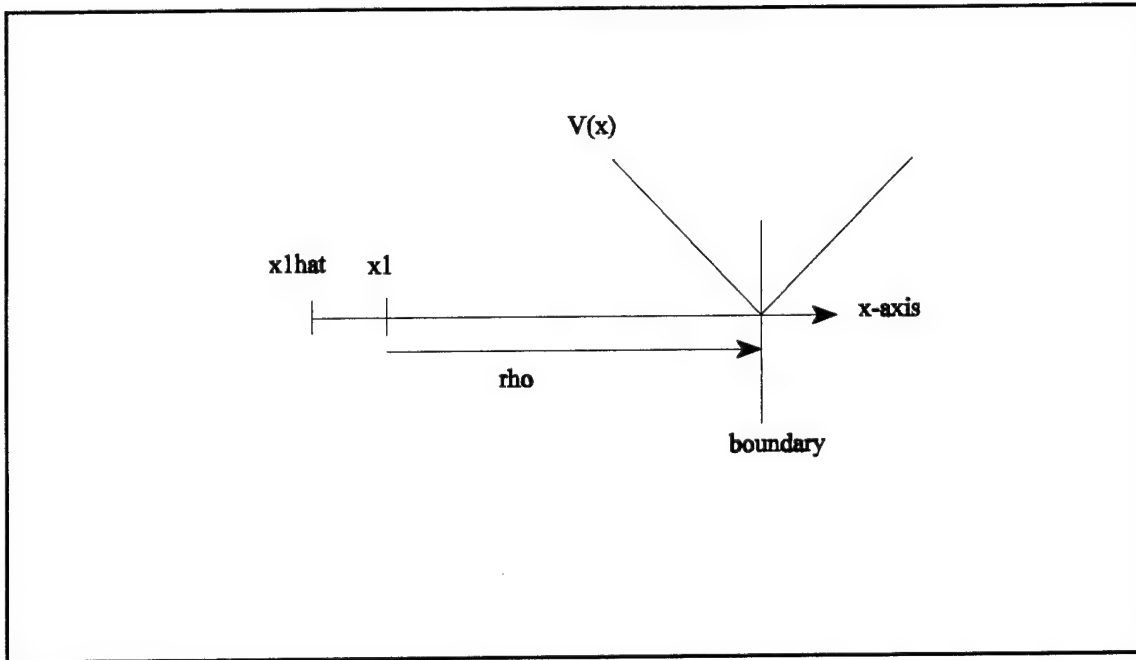
The motion model can now be represented as

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = a_x \end{cases} \quad (5.2)$$

where the states  $x_1$  and  $x_2$  are position and velocity respectively, and  $a_x$  is acceleration in the x-dimension.

In this particular case, the equations of the observer are

$$\begin{cases} \dot{\hat{x}}_1 = \hat{x}_2 - k_1 \nabla V(\hat{x}_1 + \rho) \\ \dot{\hat{x}}_2 = a_x - k_2 \nabla V(\hat{x}_1 + \rho) \end{cases} \quad (5.3)$$



**Figure 5.1 Sliding Mode Model in One Dimension**

where  $k_1$  and  $k_2$  are chosen to meet the sliding mode criteria as will be discussed shortly.

We now define the state error as

$$\begin{cases} \tilde{x}_1 = \hat{x}_1 - x_1 \\ \tilde{x}_2 = \hat{x}_2 - x_2 \end{cases} \quad (5.4)$$

and taking the time derivative of Equation (5.4) yields the error rate equations



$$\begin{cases} \dot{\tilde{x}}_1 = \dot{\hat{x}}_1 - \dot{x}_1 \\ \dot{\tilde{x}}_2 = \dot{\hat{x}}_2 - \dot{x}_2 \end{cases} \quad (5.5)$$

which combined with Equations (5.3) and (5.5) yields the following result

$$\begin{cases} \dot{\tilde{x}}_1 = \tilde{x}_2 - k_1 \nabla V(x_1 + \tilde{x}_1 + \rho) \\ \dot{\tilde{x}}_2 = -k_2 \nabla V(x_1 + \tilde{x}_1 + \rho) \end{cases} \quad (5.6)$$

Recalling that the gradient of  $V(x) = 0$  at  $x = L_1$  and  $\pm 1$  at any other value of  $x$ , Equation (5.6) can be rewritten as

$$\begin{cases} \dot{\tilde{x}}_1 = \tilde{x}_2 - k_1 \text{sign}(\tilde{x}_1) \\ \dot{\tilde{x}}_2 = -k_2 \text{sign}(\tilde{x}_1) \end{cases} \quad (5.7)$$

which defines the first-order observer for the  $x$ -dimension with a single measurement of sonar range  $\rho$ .

The values of  $k_1$  and  $k_2$  must be selected such that the phase-plane trajectories exhibit sliding behavior. The required condition [Ref. 5] is specified as

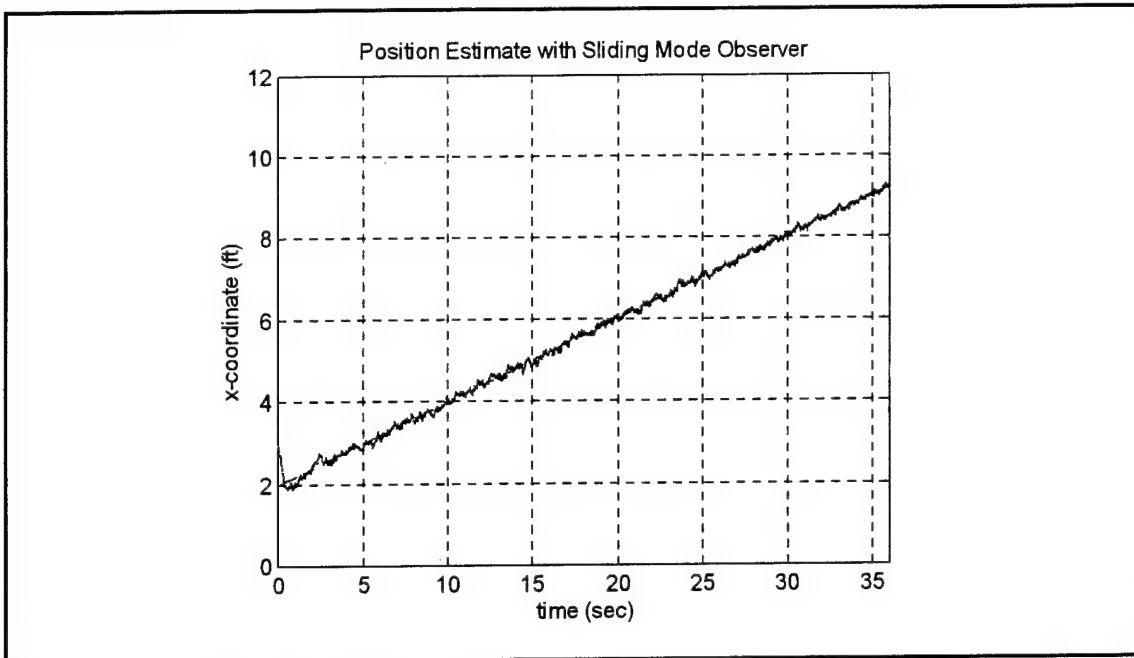
$$|x_2(0)| \leq k_1, \quad x_1 = 0 \quad (5.8)$$

which determines the value of  $k_1$ . The value of  $k_2$  is chosen for a desired rate of convergence of the state estimate.

## 2. Verification of Convergence

In order to verify satisfactory performance of the theoretical sliding observer the above algorithm was implemented for the same one-dimensional scenario addressed in Chapter II to verify convergence of the LMS algorithm. The results of the simulation are

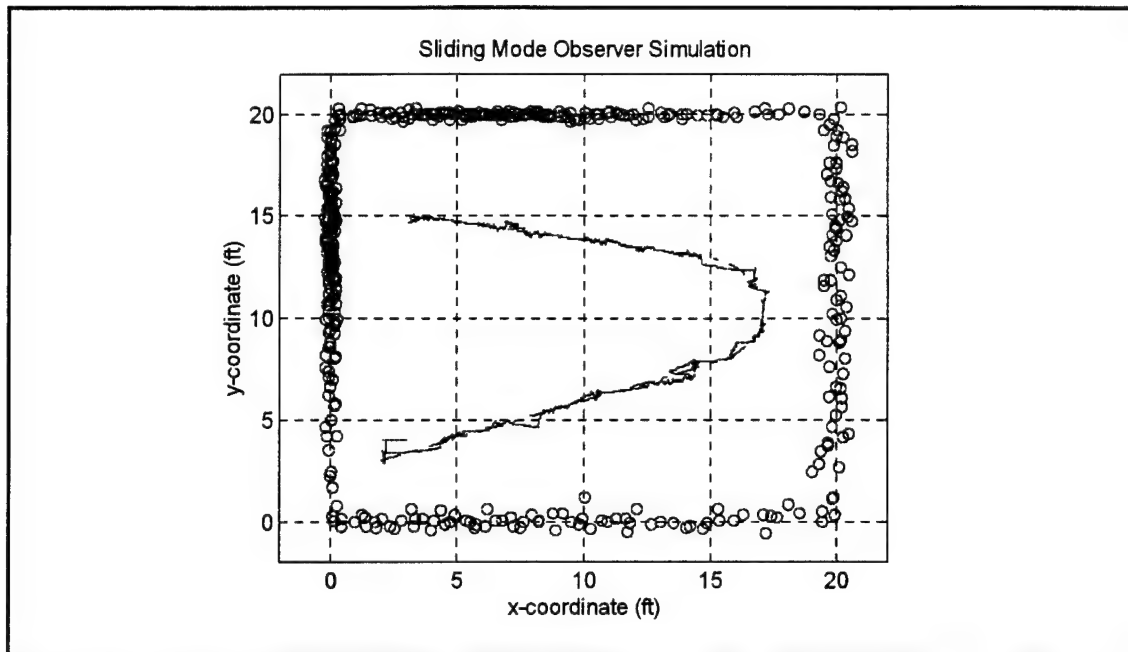
included as Figure 5.2 where it can be seen that the observer rapidly drives the position estimate trajectory to the actual trajectory of the vehicle.



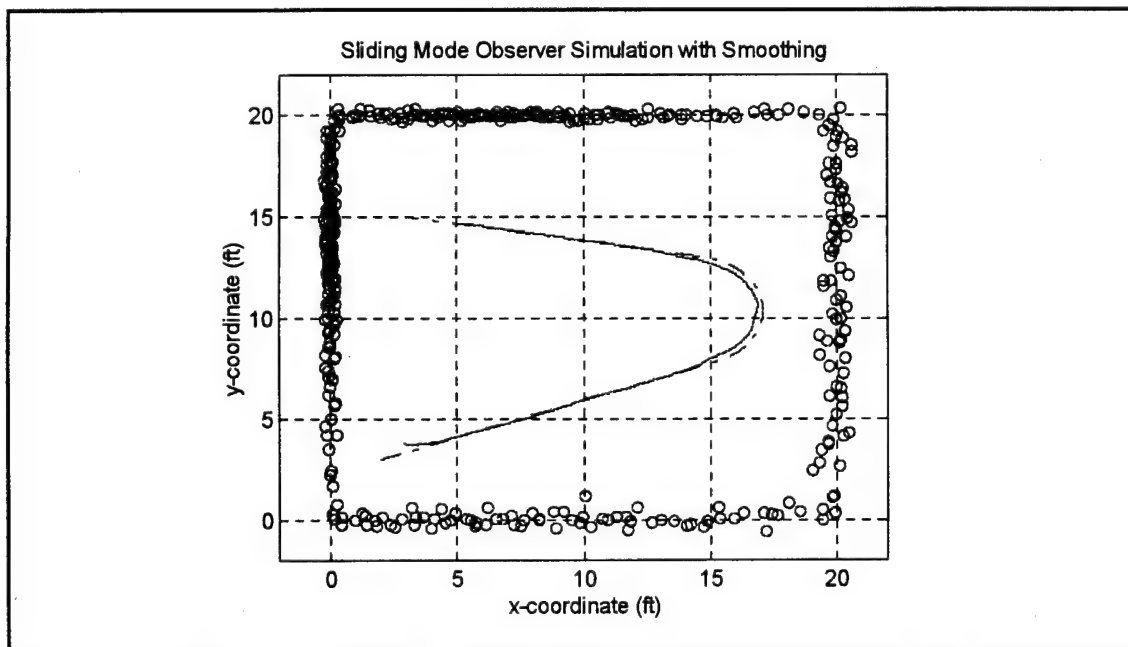
**Figure 5.2 Verification of Convergence with Sliding Observer**

The sliding mode observer theory was extended to two dimensions and utilized to simulate a closed border scenario similar to those of Chapter 2. The MATLAB code for this algorithm is included in Appendix C. Figure 5.3 shows the results of this simulation for a 72 second run consisting of eight 360-degree sonar scans of the environment.

The algorithm calculates and applies correction factors to the x-dimension position and velocity only when sonar returns are off of the vertical walls and y-dimension corrections are obtained only when the sonar beam falls on the horizontal walls. The alternate coordinate position and velocity are updated utilizing dead-reckoning of position with the sensed velocity component during the passage of the sonar beam over each side. This also causes the staircase-like appearance noted with the previous algorithm. Figure 5.4 shows the results after application of the smoothing filter.

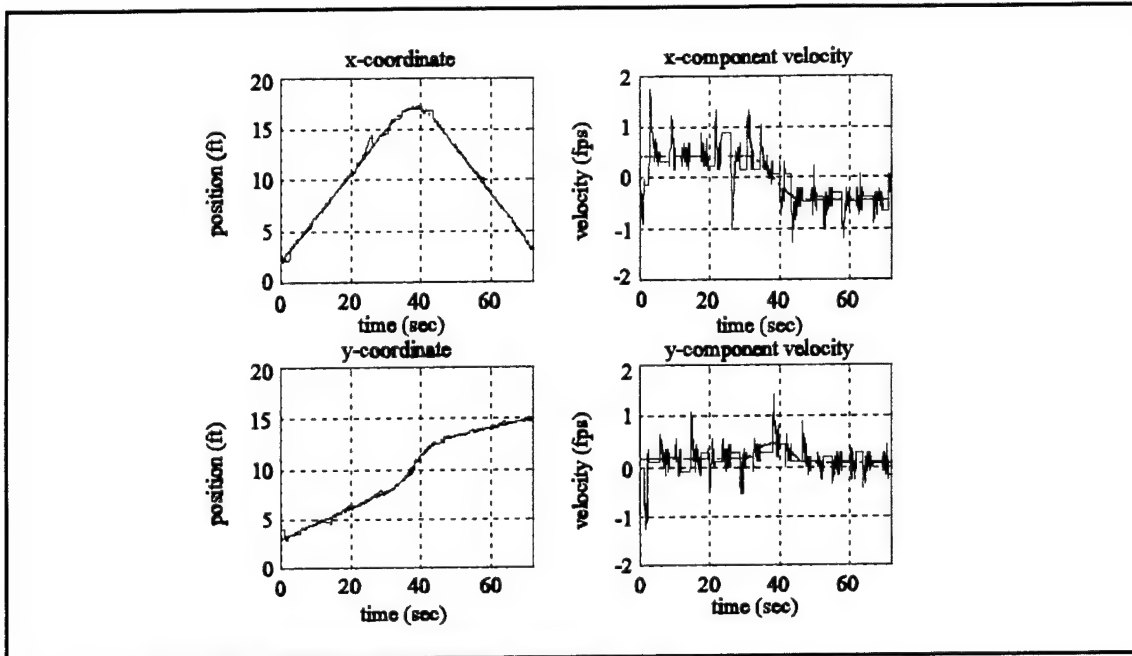


**Figure 5.3 Sliding Mode Observer Simulation**



**Figure 5.4 Sliding Mode Observer Simulation with Data Smoothing Filter**

The associated position and velocity trajectories are presented in Figure 5.5 to graphically illustrate convergence of this method in a two-dimensional application.



**Figure 5.5 Sliding Mode Observer Simulation Trajectories**

Comparison of these results to the similar scenario of Figures 3.1 and 3.2 shows that this method produces comparable results for estimating the vehicle position.

This chapter presented the initial results of an investigation into a second alternative to the Kalman Filter for vehicle state estimation. The viability of this algorithm was demonstrated for a simple one-dimensional scenario and extended to the two-dimensional closed border environment discussed in Chapter II.

## **VI. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS**

### **A. SUMMARY**

This study investigated the problem of navigating and localizing the position of an autonomous underwater vehicle in a partially known environment. A navigation algorithm was designed based on a potential function description of the operating environment combined with a Least Mean Squares (LMS) estimator. The approach taken in the design and evaluation of the navigation algorithm included:

- The development of a state-space model describing the full state of AUV motion.
- The development of a potential function description of the operating environment.
- Testing of the algorithm in scenarios of increasing complexity.
- Investigation of an alternative method utilizing a sliding mode observer as a performance comparison.

### **B. CONCLUSIONS**

The following conclusions have been reached as a result of this study:

- Estimation of vehicle position and velocity is possible with the algorithm presented.
- The potential function can adequately describe complex operating scenarios and environments.
- The algorithm performed adequately through a variety of scenarios including tests with actual sonar data gathered by an AUV.
- The algorithm is directly applicable to the NPS *Phoenix* AUV.

### C. RECOMMENDATIONS

Having demonstrated the reliability of the approach taken in this study, the following recommendations are made for follow-on research in the on-going AUV development effort at the Naval Postgraduate School:

- Convert the algorithm program to the C programming language for direct implementation into the NPS *Phoenix* control system.
- Generate sonar test data for scenarios of greater complexity to allow testing of the algorithm which goes beyond simulation/modeling.
- Develop an adaptive control strategy combining a sliding mode controller with the environment potential function to incorporate obstacle avoidance and real-time updating of the environment map.

## APPENDIX A. PARAMETER ESTIMATION

The subject of parameter estimation is addressed extensively in the technical literature. The methods utilized for this thesis follow the derivations and procedures provided in [Refs. 6 and 7].

The state-space model for the AUV addressed in Chapter 2 requires estimation of the four parameters which define the velocity and heading state differential equations

$$\begin{cases} \dot{v} = -a_1 v + b_1 u_1 \\ \ddot{\theta} = -a_2 \ddot{\theta} + b_2 u_2 \end{cases} \quad (\text{A.1})$$

where we recall that  $u_1$  and  $u_2$  represent the commanded inputs of motor rpm and rudder angle respectively.

A first-order autoregressive extended model (ARX) of the form

$$A(z^{-1})y(t) = B(z^{-1})u(t) + e(t) \quad (\text{A.2})$$

is used with  $y(t)$ ,  $u(t)$  input and output sequences, and  $e(t)$  zero mean white noise. A time delay is represented by  $z^{-1}$  and  $A$  and  $B$  are the polynomials

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \dots + a_n z^{-n} \\ B(z^{-1}) &= b_1 z^{-1} + \dots + b_n z^{-n} \end{aligned} \quad (\text{A.3})$$

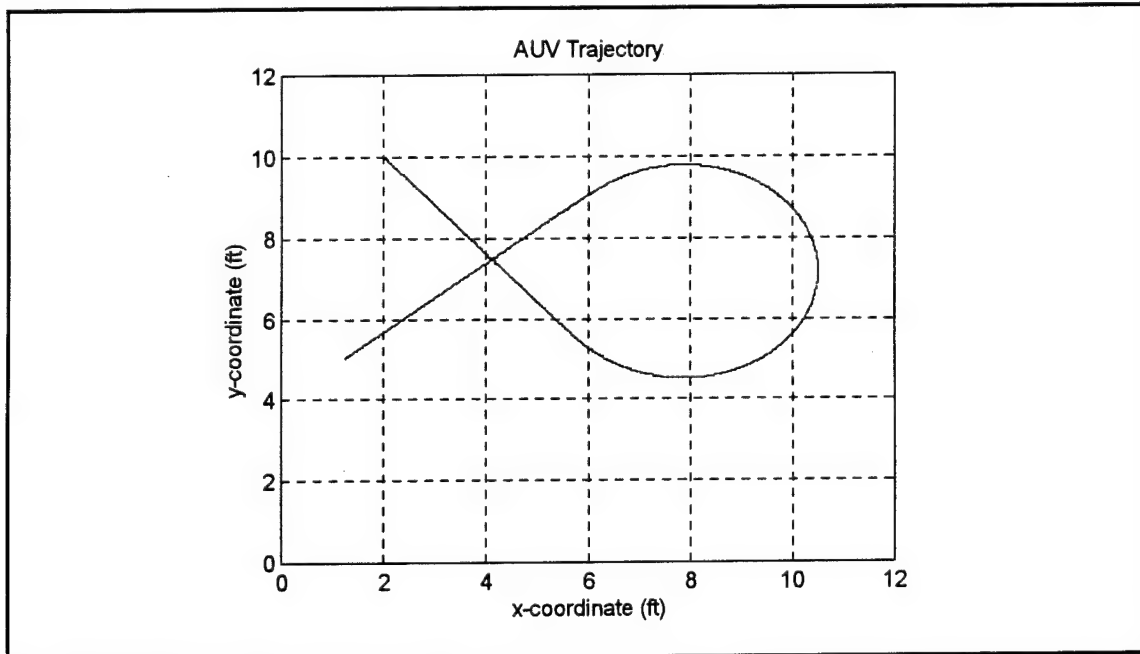
In order to estimate the  $a_i$  and  $b_i$  parameters from data sequences  $\{y\}$  and  $\{u\}$ , the model of Equation (A.2) is written in regressive form as

$$\begin{aligned}
y(t) &= \Phi^T(t) \theta + e(t) \\
\Phi^T(t) &= [-y(t-1), \dots, -y(t-n), u(t-1), \dots, u(t-n)] \\
\theta &= [a_1, \dots, a_n, b_1, \dots, b_n]
\end{aligned} \tag{A.4}$$

The recursive least-squares (RLS) equations can then be written as

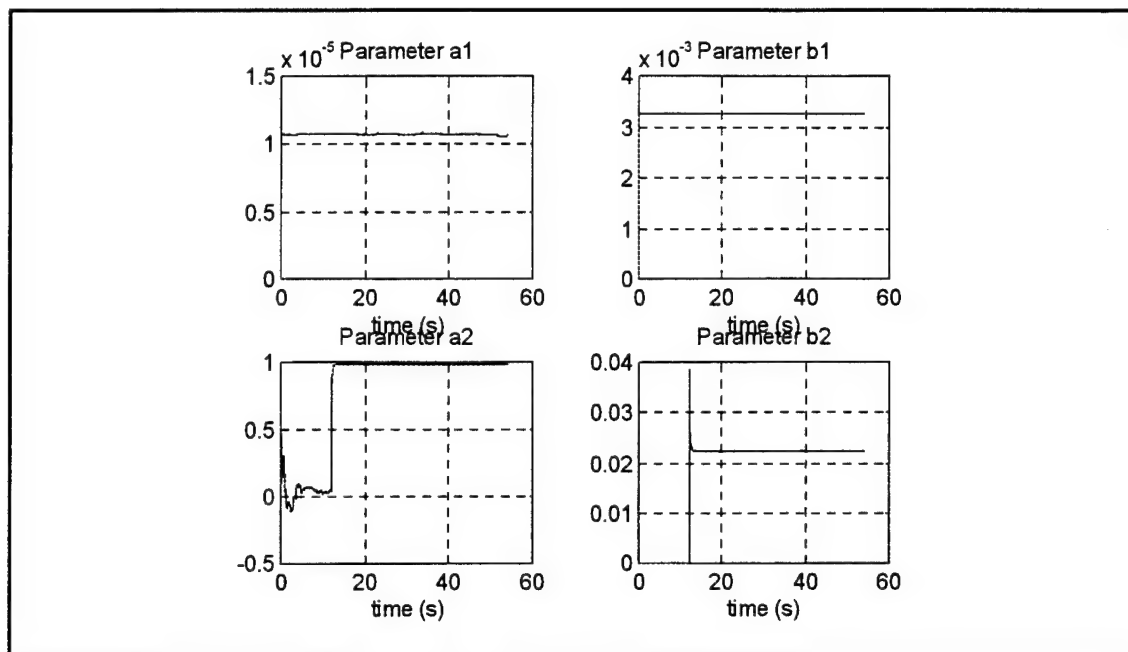
$$\begin{aligned}
\hat{\theta}(t+1) &= \hat{\theta}(t) + K(t)[y(t) - \Phi^T(t)\hat{\theta}(t)] \\
K(t) &= P(t)\Phi(t)[1 + \Phi^T(t)P(t)\Phi(t)]^{-1} \\
P(t+1) &= P(t) - P(t)\Phi(t)\Phi^T(t)P(t)[1 + \Phi^T(t)P(t)\Phi(t)]^{-1}
\end{aligned} \tag{A.5}$$

Vehicle data obtained from the AUV trajectory of Figure A.1 was utilized to estimate the system parameters of the state-space model. The MATLAB script file PARAMEST.m implements the RLS algorithm and produced the parameter trajectories of Figure A.2.



**Figure A.1 AUV Trajectory for Parameter Estimation**





**Figure A.2 Discrete Time System Parameters**

Conversion of the steady-state discrete time parameters results in the following values utilized for the continuous-time nonlinear state space model parameters:

$$\begin{aligned}
 a_1 &= 508.4 \\
 b_1 &= 1.667 \\
 a_2 &= 1 \\
 b_2 &= 1
 \end{aligned}
 \tag{A.6}$$

```

%%%%%%%% File PARAMEST.m
%%%%%%%%
%%%%%%%% This script file estimates the state-space model
%%%%%%%% parameters utilizing a Recursive Least-Squares algorithm
%%%%%%%% based on an input data file containing velocity and heading
%%%%%%%% data for the NPS Phoenix AUV.
%%%%%%%%

load prmsdata      % Load data file
time=prmsdata(:,1); % Time (sec)
v=prmsdata(:,2);   % Velocity (ft/s)
hdg=prmsdata(:,3); % Heading Rate (deg/s)
rpm=prmsdata(:,4); % Motor Command (rpm)
rud=prmsdata(:,5); % Rudder Angle (deg)
n=length(v);
Ts=0.0225;         % Specify Time Step
% Estimate Velocity Parameters with First Order ARX Model
th1=zeros(2,n);    % Initialize Output Matrix
P1=1000*eye(2);    % Initialize Error Covariance
for i=2:n-1
    phi1(i,:)=[-v(i-1) rpm(i-1)]; % Data Input
% Calculate Gain
    K1(:,i)=P1*phi1(i,:)'*inv(1+phi1(i,:)*P1*phi1(i,:));
% Update Covariance Matrix
    P1=P1-P1*phi1(i,:)'*phi1(i,:)*P1*inv(1+phi1(i,:)*P1*phi1(i,:));
% Parameter Estimate Update
    th1(:,i+1)=th1(:,i)+K1(:,i)*(v(i)-phi1(i,:)*th1(:,i));
end

```

```

% Estimate Heading Parameters
th2=zeros(2,n);           % Initialize Output Matrix
P2=1000*eye(2);           % Initialize Error Covariance
for i=2:n-1
    phi2(i,:)=[-hdg(i-1) rud(i-1)];    % Data Input
% Calculate Gain
    K2(:,i)=P2*phi2(i,:)'*inv(1+phi2(i,:)*P2*phi2(i,:)');
% Update Covariance Matrix
    P2=P2-P2*phi2(i,:)'*phi2(i,:)*P2*inv(1+phi2(i,:)*P2*phi2(i,:)');
% Parameter Estimate Update
    th2(:,i+1)=th2(:,i)+K2(:,i)*(hdg(i)-phi2(i,:)*th2(:,i));
end
% Convert Discrete Time Parameters to Continuous Parameters
phi1=[-th1(1,n) 0;0 -th2(1,n)];
del1=[th1(2,n) 0;0 th2(2,n)];
[A1,B1]=d2c(phi1,del1,Ts);
prms=[-A1(1,1);B1(1,1);-A1(2,2);B1(2,2)];
% Plot Discrete Time Parameters
clg
subplot(221)
plot(time,-th1(1,:),'b'); xlabel('time (s)'); title('Parameter a1'); grid
subplot(222)
plot(time,th1(2,:),'b'); xlabel('time (s)'); title('Parameter a2'); grid
subplot(223)
plot(time,-th2(1,:),'b'); xlabel('time (s)'); title('Parameter b1'); grid
subplot(224)
plot(time,th2(2,:),'b'); xlabel('time (s)'); title('Parameter b2'); grid

```



## APPENDIX B. SONAR CHARACTERISTICS

The primary sonar currently installed aboard the NPS *Phoenix* AUV is the Tritech ST1000 Profiling System. This profiling sonar is a high performance, high resolution compact and lightweight system which produces accurate underwater measurements. The following list provides a summary of the sonar performance characteristics and specifications:

Frequency .....	1 MHz
Beamwidth .....	0.9 degree conical
Maximum Range .....	50 meters (160 feet)
Minimum Range .....	0.3 meters (1 foot)
Weight .....	1.5 kg
View Sector .....	360 degrees
Scan Rate .....	40 degrees/second

These parameters were utilized for producing simulated sonar data in all theoretical simulations conducted for this thesis.



## APPENDIX C. MATLAB AND SIMULINK FILES

### A. GENERAL

This appendix contains the files generated utilizing the MATLAB with SIMULINK software package. The programs are separated by section for each Chapter of the thesis main body. Each program is headed with a short description of its function and usage and includes appropriate comments for documentation. In general, each type of scenario is simulated with a controlling program named MAIN\*\*\*\*.m with the asterisks replaced by a descriptive abbreviation such as "OPEN" for an open environment simulation. All functions which calculate sonar data are named SONAR\*\*\*\*.m, potential function calculation programs are listed as VG\*\*\*\*.m, etc. with the asterisks being the same for all associated subprograms.

### B. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER II

```
%%%%%%%% MATLAB Script File CONVTEST.m is utilized to verify the convergence
%%%%%%%% of the navigation algorithm in a simple one-dimensional scenario
%%%%%%%% for two cases of 1) Velocity measured and 2) Acceleration measured.
%%%%%%%%
```

```
% Specify Simulation Parameters
```

```
Ts=0.0225;          % Step Size
```

```
time=Ts*(0:1600);  % Generate time vector
```

```
xact=3:0.2*Ts:10.2; % Generate actual vehicle position
```

```
lamda=0.25;        % Describe potential function
```

```
% Case 1: Measured Velocity
```

```
mul=10;             % Position rate estimation gain
```

```
vact=0.2;           % Actual vehicle velocity (fps)
```

```
xest1=zeros(1,1601); % Initialize output vector
```

```

xest1(1)=4;           % Initial position estimate (ft)
phatdot=zeros(1,1601); % Initialize position rate estimate vector
% Iterative Algorithm
for k=2:1601
    s=(12-xact(k-1))*(1+0.05*randn(1,1)); % Sonar data w/ noise
    vwn=vact*(1+0.05*randn(1,1)); % Velocity measurement w/ noise
    shat=xest1(k-1)+s; % Position of tip of sonar vector
    [v,dvx]=vgtest(shat,lamda); % Calculate value of potential fcn
    phatdot(k)=vwn+mu1*v*dvx; % Estimate position rate
    xest1(k)=xest1(k-1)+Ts*phatdot(k); % Update position estimate
end
% Generate Output
clg
plot(time,xact,'b-.')
grid
hold
plot(time,xest1,'b')
axis([0 36 0 12])
xlabel('time (sec)')
ylabel('x-coordinate (ft)')
title('Position Estimate w/ Measured Velocity')
pause
% Case 2: Measured Acceleration
mu2=10; % Velocity rate estimation gain
xest2=zeros(1,1601); % Initialize output vector
xest2(1)=4; % Initial position estimate (ft)
a=0; % Vehicle Acceleration (ft/s^2)
vhat=zeros(1,1601); % Initialize velocity estimate vector

```



```

vhat(1)=0;           % Initial velocity estimate (fps)
vhatdot=zeros(1,1601); % Initialize velocity rate estimate vector
% Iterative Algorithm
for n=2:1601
    s=(12-xact(n-1))*(1+0.05*randn(1,1)); % Sonar data w/noise
    shat=xest2(n-1)+s; % Position of tip of sonar vector
    [v,dvx]=vgtest(shat,lamda); % Calculate value of potential fcn
    vhatdot(n)=a+mu2*v*dvx; % Estimate velocity rate
    vhat(n)=vhat(n-1)+Ts*vhatdot(n); % Update velocity estimate
    phatdot(n)=vhat(n)+mu1*v*dvx; % Estimate position rate
    xest2(n)=xest2(n-1)+Ts*phatdot(n); % Update position estimate
end
% Generate Output
clg
subplot(211)
plot(time,xact,'b-.')
grid
hold
plot(time,xest2,'b')
axis([0 36 0 12])
xlabel('time (sec)')
ylabel('x-coordinate (ft)')
title('Position Estimate w/ Measured Acceleration')
subplot(212)
plot(time,vact*ones(1,1601),'b-.')
grid
hold
plot(time,vhat,'b')

```

```
axis([0 36 -0.4 0.4])
xlabel('time (sec)')
ylabel('velocity (fps)')
```

```
%%%%%%%% MATLAB script file CORRCALC.m assists the user in determining an
%%%%%%%% an appropriate value for both lamda and mu by showing the value
%%%%%%%% of the navigation correction factor and the width of the attractive
%%%%%%%% field in either the x or y-axis direction.
%%%%%%%%
```

```
% Prompt User for Input Data
```

```
lamda=input('Enter value of lamda: ');
mu=input('Enter value of mu: ');
flag=input('Enter "1" if x-correction desired or "2" if y-correction: ');
if flag ==1
    val=input('Enter y-coordinate (ft): ');
else
    val=input('Enter x-coordinate (ft): ');
end
begin=input('Enter start of range of interest (ft): ');
stop=input('Enter end of range of interest (ft): ');
range=begin:0.02:stop;
long=length(range);
% Calculate Correction Factors
for k=1:long
    if flag == 1
        [v,dvx,dvy]=vgrad(range(k),val,lamda);
        corr(k)=mu*v*dvx;
```

```

else
    [v,dvx,dvy]=vgrad(val,range(k),lamda);
    corr(k)=mu*v*dvy;
end
end
% Plotting Commands
plot(range,corr)
xlabel('feet')
ylabel('magnitude')
title('Navigation Correction Factor Magnitude (One-dimension)')

```

### C. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER III

```

%%%%%  MATLAB File AUVPOSIT.M prompts the user for simulation parameters
%%%%%  and then utilizes a state-space model to determine actual AUV
%%%%%  data for the length of the simulation. The output state vector
%%%%%  consists of x-coordinate, y-coordinate, velocity, hdg (in degrees),
%%%%%  and heading rate.

```

```

% Prompt user for simulation parameters
Tf=input('Enter length of simulation (seconds): ');
x0=input('Enter initial x-coordinate (ft): ');
y0=input('Enter initial y-coordinate (ft): ');
theta0=input('Enter initial course (degrees): ');
turns=input('Enter time to start turn (seconds): ');
rudang=input('Enter rudder angle (degrees)(+ left, - right): ');
turne=input('Enter time to stop turn (seconds): ');
v0=input('Enter initial vehicle velocity (ft/s): ');

```

```

rpm=input('Enter commanded motor rpm: ');
L1=input('Enter x-coordinate range of oparea (ft): ');
L2=input('Enter y-coordinate range of oparea (ft): ');
disp(' ');
Ts=0.0225;           % Time Step
k=round(Tf/Ts)+1;     % Number of Iterations
state=zeros(5,k-1);   % Initialize State Vector (Matrix)
state(:,1)=[x0 y0 v0 theta0 0]'; % Initialize State Vector
hdg(1)=state(4,1);
u=[rpm*ones(1,k);zeros(1,k)]; % Initialize Control Input
indexs=round(turns/Ts); % Index when Turn Starts
indexe=round(turne/Ts); % Index when Turn Ends
tlength=indexe-indexs+1; % Length of Turn in # Steps
tu=rudang*ones(1,tlength); % Rudder Control Input During Turn
u(2,indexs:indexe)=tu;
% Define Coefficients for Time-Dependent State-Space Model
alpha=508.4;
beta=1.6677;
gamma=1;
delta=1;
% Run State-Space Model for Length of Simulation
for i=2:k
    a=cos(state(4,i-1)*pi/180);
    b=sin(state(4,i-1)*pi/180);
    A=[0 0 a 0 0;0 0 b 0 0;0 0 -alpha 0 0;0 0 0 0 1;0 0 0 0 -gamma];
    B=[0 0;0 0;beta 0;0 0;0 delta];
    [phi,del]=c2d(A,B,Ts);
    state(:,i)=phi*state(:,i-1)+del*u(:,i-1);

```

```

    hdg(i)=state(4,i);
    if hdg(i)<0
        hdg(i)=hdg(i)+360;
    end
end
% Resolve Velocity into xy-components
xvel=state(3,:).*(cos((pi/180)*state(4,:)));
yvel=state(3,:).*(sin((pi/180)*state(4,:)));
% Output/Plotting Commands
timvec=0:Ts:Tf;
orient landscape
clg
subplot(211)
plot(timvec,state(1,:))
axis([0 Tf 0 L1])
grid
title('x-coordinate vs. time')
xlabel('time - sec')
ylabel('feet')
subplot(212)
plot(timvec,state(2,:))
axis([0 Tf 0 L2])
grid
title('y-coordinate vs. time')
xlabel('time - sec')
ylabel('feet')
pause
clg

```

```

subplot(211)
plot(timvec,state(3,:))
axis([0 Tf 0 0.6])
grid
title('Velocity vs. Time')
xlabel('time - sec')
ylabel('ft/sec')
subplot(212)
plot(timvec,hdg)
axis([0 Tf 0 360])
grid
title('Heading vs. Time (Angle referenced to x-axis)')
xlabel('time - sec')
ylabel('degrees')
pause
clg
subplot(211)
plot(timvec,state(5,:))
axis([0 Tf -12 12])
grid
title('Heading Rate vs. Time (ccw = positive)')
xlabel('time - sec')
ylabel('degrees/sec')
pause
clg
subplot(211)
plot(timvec,u(1,:))
axis([0 Tf 0 150])

```

```

grid
title('Speed (rpm) Input vs. Time')
xlabel('time - sec')
ylabel('rpm')
subplot(212)
plot(timvec,u(2,:))
axis([0 Tf -12 12])
grid
title('Commanded Rudder Angle Input vs. Time')
xlabel('time - sec')
ylabel('degrees')
pause
orient portrait
clg
plot(state(1,:),state(2,:))
axis([0 L1 0 L2])
grid
title('AUV Actual Track')
xlabel('x-coordinate - feet')
ylabel('y-coordinate - feet')
pause

```

```

%%%%%      MATLAB File MAINSQ.M is the main program which prompts
%%%%%      the user for the parameters to be utilized in the navigation
%%%%%      algorithm and the initial position estimate. This program
%%%%%      calls several supporting programs to process the sonar
%%%%%      measurements and calculate the value and gradient of the potential
%%%%%      function.

```

```

% Prompt user for simulation parameters
lamda=input('Enter value of lamda (50 recommended): ');
mu=input('Enter value of mu (5 recommended): ');
xhat0=input('Enter initial estimate of x-coordinate: ');
yhat0=input('Enter initial estimate of y-coordinate: ');
L1=input('Enter x-dimension width of oparea (ft): ');
L2=input('Enter y-dimension height of oparea (ft): ');
disp(' ');

% Initialize algorithm
global L1 L2
phat0=[xhat0;yhat0];           % Initial Position Estimate
phat=zeros(2,Tf/9*400);       % Initialize Output Position Matrix
theta=0;                       % Initial Sonar Bearing
phat(:,1)=phat0;
s0=sonar(state(1,1),state(2,1),theta); % Initial Sonar Vector
temps=phat0+s0;
border(:,1)=temps;            % Initial Pool Border Estimate

% Run Simulation
for n=2:k
    x=state(1,n);              % Actual x-coordinate of AUV
    y=state(2,n);              % Actual y-coordinate of AUV
    s=sonar(x,y,theta);        % Noise-corrupted Sonar Vector
    z0=[state(3,n);state(4,n)*pi/180]; % Vehicle Velocity & Heading
    zcurr=z0+[randn(1)*0.05*z0(1);randn(1)*0.5*pi/180]; % w/ Error
    pdot=[zcurr(1)*cos(zcurr(2));zcurr(1)*sin(zcurr(2))]; % vx,vy
    shat=phat(:,n-1)+pdot*Ts+s; % Estimated Position + Sonar Vector
    border(:,n)=shat;          % Estimated Border Point of Pool
    % Estimate of Pool Border Orientation

```



```

sangle=atan(abs(shat(2)-temps(2))/abs(shat(1)-temps(1)));
temps=shat;
% Calculate Value and Gradient of Potential Function
[v,dvx,dvy]=vgrad(shat(1),shat(2),lamda);
% Apply Correction Factor to Update x and y velocity estimates
pdot=dpdot+mu*v*[dvx*sin(sangle);dvy*cos(sangle)];
% Obtain current position estimate
phat(:,n)=phat(:,n-1)+Ts*pdot;
theta=theta+0.9*pi/180;    % Increment Sonar Head Bearing
if theta >=2*pi           % Reset theta to 0 after 360 deg
    theta=0;
end
end
end

```

```

%%%%%      MATLAB Function VGRAD.M calculates the value of the
%%%%%%%%   potential function and its gradient for a given input
%%%%%%%%   position and the parameter lamda.
%%%%%%%%

```

```

function [v,dvx,dvy]=vgrad(x,y,lamda);
global L1 L2
v0=(x.*(x-L1))*(y.*(y-L2));    % Value of Potential Function
z=v0/lamda;
v=sigmoid(z);                  % Apply "Squashing" Function
dvx=-dsig(z)*((2*x-L1)*(y.*(y-L2)))/lamda; % x-component of gradient
dvy=-dsig(z)*((x.*(x-L1))*(2*y-L2))/lamda; % y-component of gradient
end

```

```

%%%%%%%% MATLAB Function SIGMOID.M calculates the value of the
%%%%%%%% sigmoid utilized as a squashing function.
%%%%%%%%

```

```

function y=sigmoid(x)
    x=min(x,100);
    x=max(x,-100);
    y=(1-exp(-x))./(1+exp(-x));
end

```

```

%%%%%%%% MATLAB Function DSIG.M calculates the derivative of the
%%%%%%%% sigmoid function.
%%%%%%%%

```

```

function d=dsig(x)
    x=min(x,100);
    x=max(x,-100);
    temp=exp(-x);
    d=2*temp./(1+2*temp+temp.*temp);
end

```

```

%%%%%%%% MATLAB File WITHDATA.m commands the navigation algorithm when
%%%%%%%% processing actual test data obtained with the NPS PHOENIX AUV.
%%%%%%%% The sonar test data is loaded into the MATLAB Workspace as a
%%%%%%%% matrix variable "data" consisting of column vectors of bearing
%%%%%%%% and range. The test data in ASCII named rob1.d, etc. consists of
%%%%%%%% measurements taken at 0.045 second intervals requiring a modification
%%%%%%%% to the regular algorithm integration step size.

```

```

% Prompt user for simulation parameters
lamda=input('Enter value of lamda: ');
xhat0=input('Enter value of xhat0: ');
yhat0=input('Enter value of yhat0: ');
mu=input('Enter value of mu: ');
Ts=0.045; % Integration step size
% Initialize algorithm
phat0=[xhat0;yhat0]; % Initial position estimate
phat=zeros(2,1201); % Initialize output position est
border=zeros(2,1201); % Initialize border
% Run Simulation
s0=[data(1,1);data(1,2)]; % Initial sonar measurement
temps=phat0+[s0(2)*cos(s0(1)*pi/180);s0(2)*sin(s0(1)*pi/180)];
border(:,1)=temps;
phat(:,1)=phat0;
for n=2:1201
    if n >= 1000 % If reqd increase fcn width
        lamda=4000;
        mu=400;
    end
    s=[data(n,1);data(n,2)]; % Current sonar measurement
    shat=phat(:,n-1)+[s(2)*cos(s(1)*pi/180);s(2)*sin(s(1)*pi/180)];
    border(:,n)=shat;
    % Estimate of Pool Border Orientation
    sangle=atan(abs(shat(2)-temps(2))/abs(shat(1)-temps(1)+1e-9));
    temps=shat;
    % Calculate Value and Gradient of Potential Function
    [v,dvx,dvy]=vgrad6(shat(1),shat(2),lamda);

```

```

% Apply Correction Factor to Update x and y velocity estimates
pdot=pdot+mu*v*[dvx*sin(sangle);dvy*cos(sangle)];
% Obtain current position estimate
phat(:,n)=phat(:,n-1)+Ts*pdot;
end

```

```

%%%%%%%% MATLAB Function VGRAD.M calculates the value of the
%%%%%%%% potential function and its gradient for a given input
%%%%%%%% position and the parameter lamda for simulations with
%%%%%%%% actual sonar test data in the 6x6 meter test pool.
%%%%%%%%

```

```

function [v,dvx,dvy]=vgrad(x,y,lamda);
v0=(x.*(x-19.68))*(y.*(y-19.68))'; % Value of Potential Function
z=v0/lamda;
v=sigmoid(z); % Apply "Squashing" Function
dvx=-dsig(z)*((2*x-19.68)*(y.*(y-19.68)))/lamda; % x-component of gradient
dvy=-dsig(z)*((x.*(x-19.68))*(2*y-19.68))/lamda; % y-component of gradient
end

```

```

%%%%%%%% MATLAB File MAINOBS1.m is the main program which prompts
%%%%%%%% the user for the parameters to be utilized in the navigation
%%%%%%%% algorithm and the initial position estimate for a scenario including
%%%%%%%% a cylindrical obstacle modeled in the potential function. This
%%%%%%%% program calls several supporting programs to process the sonar
%%%%%%%% measurements and calculate the value and gradient of the potential
%%%%%%%% function.
%%%%%%%%

```

```

% Prompt user for simulation parameters
lamda=input('Enter value of lamda: ');
mu=input('Enter value of mu: ');
xhat0=input('Enter initial estimate of x-coordinate: ');
yhat0=input('Enter initial estimate of y-coordinate: ');
disp(' ');

% Initialize algorithm
phat0=[xhat0;yhat0]; % Initial Position Estimate
phat=zeros(2,Tf/9*400); % Initialize Output Position Matrix
theta=0; % Initial Sonar Bearing
phat(:,1)=phat0;
s0=sonobs(state(1,1),state(2,1),theta); % Initial Sonar Vector
temps=phat0+[s0(1);s0(2)];
border(:,1)=temps; % Initial Pool Border Estimate

% Run Simulation
for n=2:k
    x=state(1,n); % Actual x-coordinate of AUV
    y=state(2,n); % Actual y-coordinate of AUV
    s=sonobs(x,y,theta); % Noise-corrupted Sonar Vector
    z0=[state(3,n);state(4,n)*pi/180]; % Vehicle Velocity & Heading
    zcurr=z0+[randn(1)*0.05*z0(1);randn(1)*0.5*pi/180]; % w/ Error
    pdot=[zcurr(1)*cos(zcurr(2));zcurr(1)*sin(zcurr(2))]; % vx,vy
    shat=phat(:,n-1)+pdot*Ts+[s(1);s(2)]; % Estimated Position + Sonar Vector
    border(:,n)=shat; % Estimated Border Point of Pool

    % Estimate of Pool Border Orientation
    if s(3)==1
        sangle1=pi/2;
        sangle2=0;
    end
end

```

```

else
    sangle1=atan(abs(shat(2)-temps(2))/abs(shat(1)-temps(1)));
    sangle2=sangle1;
end
temps=shat;
% Calculate Value and Gradient of Potential Function
[v,dvx,dvy]=vgcyl(shat(1),shat(2),lamda);
% Apply Correction Factor to Update x and y velocity estimates
pdotat=pdot+mu*v*(lamda/lamda0)*[dvx*sin(sangle1);dvy*cos(sangle2)];
% Obtain current position estimate
phat(:,n)=phat(:,n-1)+Ts*pdotat;
theta=theta+0.9*pi/180;           % Increment Sonar Head Bearing
if theta >=2*pi                   % Reset theta to 0 after 360 deg
    theta=0;
end
end
end

```

```

%%%%%%%%%      MATLAB Function SONAR.M calculates the sonar range and
%%%%%%%%%      bearing to the rectangular pool border given the position in
%%%%%%%%%      the pool and the angle of the sonar head. The bearing
%%%%%%%%%      and range are corrupted with noise (error). The environment
%%%%%%%%%      includes one cylindrical obstacle.
%%%%%%%%%

```

```

function s = sonobs(x,y,theta);
global yo                % Obstacle Description
global xo
global r

```

```

flag=0;
% Calculate angles from position to 4 pool corners
theta1=atan2(12-y,12-x);      % Upper Right Corner
theta2=atan2(x,12-y)+pi/2;     % Upper Left Corner
theta3=atan2(y,x)+pi;          % Lower Left Corner
theta4=atan2(12-x,y)+3*(pi/2); % Lower Right Corner
% Calculate angle to obstacle
thetobs=atan2(yo-y,xo-x);
if thetobs <= 0
    thetobs=2*pi+thetobs;
end
robs=sqrt((xo-x)^2+(yo-y)^2);
diff=atan(r/robs);
if (theta >= (thetobs-diff)) & (theta <= (thetobs+diff))
    flag=1;
    rho=robs-r;
else
    if theta <= theta1          % Calculate Range
        rho=(12-x)/cos(theta);
    elseif theta <= theta2
        rho=(12-y)/sin(theta);
    elseif theta <= theta3
        rho=x/cos(pi-theta);
    elseif theta <= theta4
        rho=y/abs(cos(1.5*pi-theta));
    else
        rho=(12-x)/cos(theta);
    end
end

```

```

end
rhon=rho*(1+0.0225*randn(1,1));           % Add Range Error
thetan=theta+(0.45*pi/180)*randn(1,1);    % Add Bearing Error
s=[rhon*cos(thetan);rhon*sin(thetan);flag]; % Convert to x,y Components
end

%%%%%%%% MATLAB Function VGCYL.M calculates the value of the
%%%%%%%% potential function and its gradient for a given input
%%%%%%%% position and the parameter lamda for the rectangular
%%%%%%%% environment with one modelled obstacle.

function [v,dvx,dvy]=vgcyl(x,y,lamda);

global xo yo r           % Description of Obstacle
a=xo;
b=yo;
c=r^2;
v0=(x.*(x-12).*(x-a).^2)*(y.*(y-12))'+(x.*(x-12))*(y.*(y-12).*(y-b).^2)'-c*(x.*(x-12)).
*(y.*(y-12))'; % Value of Potential Function
z=v0/lamda;
v=sigmoid(z);           % Apply "Squashing" Function
dvx=-dsig(z)*(((x-12).*(x-a).^2)*(y.*(y-12))'+(x.*(x-a).^2)*(y.*(y-12))'+(2*x.*(x-12).
*(x-a))*(y.*(y-12))'+(x-12)*(y.*(y-12).*(y-b).^2)' +x*(y.*(y-12).*(y-b).^2)'-c*(x-12)*
(y.*(y-12))'-c*x*(y.*(y-12)))'/lamda; % x-component of gradient
dvy=-dsig(z)*((x.*(x-12).*(x-a).^2)*(y-12)'+(x.*(x-12).*(x-a).^2)*y'+(x.*(x-12))*
((y-12).*(y-b).^2)'+(x.*(x-12))*(y.*(y-b).^2)'+(x.*(x-12))*(2*y.*(y-12).*(y-b))'-c*
(x.*(x-12))*(y-12)'-c*(x.*(x-12))*y')/lamda; % y-component of gradient
end

```



#### D. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER IV

```
%%%%%%%% MATLAB File MAINOPEN.m implements the navigation algorithm in
%%%%%%%% the open environment scenario. It prompts
%%%%%%%% the user for the parameters to be utilized in the navigation
%%%%%%%% algorithm and the initial position estimate (guess). This program
%%%%%%%% calls several supporting programs to process the sonar
%%%%%%%% measurements and calculate the value and gradient of the potential
%%%%%%%% function.
```

```
% Prompt user for simulation parameters
```

```
lamda=input('Enter value of lamda (50 recommended): ');
```

```
mu=input('Enter value of mu (10 recommended): ');
```

```
xhat0=input('Enter initial estimate of x-coordinate (ft): ');
```

```
yhat0=input('Enter initial estimate of y-coordinate (ft): ');
```

```
disp(' ');
```

```
% Initialize algorithm
```

```
phat0=[xhat0;yhat0]; % Initial Position Estimate
```

```
phat=zeros(2,Tf/9*400); % Initialize Output Position Matrix
```

```
theta=0; % Initial Sonar Bearing
```

```
phat(:,1)=phat0;
```

```
s0=sonopen2(state(1,1),state(2,1),theta); % Initial Sonar Vector
```

```
temps=phat0+[s0(1);s0(2)];
```

```
border(:,1)=temps; % Initial Pool Border Estimate
```

```
% Run Simulation
```

```
for n=2:k
```

```
    x=state(1,n); % Actual x-coordinate of AUV
```

```
    y=state(2,n); % Actual y-coordinate of AUV
```

```

s=sonopen2(x,y,theta);           % Noise-corrupted Sonar Vector
z0=[state(3,n);state(4,n)*pi/180]; % Vehicle Velocity & Heading
zcurr=z0+[randn(1)*0.05*z0(1);randn(1)*0.5*pi/180]; % w/ Error
pdot=[zcurr(1)*cos(zcurr(2));zcurr(1)*sin(zcurr(2))]; % vx,vy
shat=phat(:,n-1)+pdot*Ts+[s(1);s(2)]; % Estimated Position + Sonar Vector
border(:,n)=shat;                % Estimated Border Point of Pool
temps=shat;
% Calculate Value and Gradient of Potential Function
[v,dvx,dvy]=vgopen(shat(1),shat(2),lamda);
% Apply Correction Factor to Update x and y velocity estimates
pdothat=pdot+mu*v*[dvx,dvy];
% Obtain current position estimate
phat(:,n)=phat(:,n-1)+Ts*pdothat;
theta=theta+0.9*pi/180;          % Increment Sonar Head Bearing
if theta >=2*pi                  % Reset theta to 0 after 360 deg
    theta=0;
end
end

%%%%%      MATLAB Function SONAR.M calculates the sonar range and
%%%%%      bearing for the open environment where navigation corrections
%%%%%      are derived from two markers (obstacles) place in the environment.
%%%%%

function s = sonobs(x,y,theta);
global xo1 yo1 xo2 yo2 r1 r2      % Description of obstacles
% Calculate angles to obstacles
thetobs1=atan2(yo1-y,xo1-x);

```

```

thetobs2=atan2(yo2-y,xo2-x);
if thetobs1 <= 0
    thetobs1=2*pi+thetobs1;
end
if thetobs2 <= 0
    thetobs2=2*pi+thetobs2;
end
robs1=sqrt((xo1-x)^2+(yo1-y)^2);           % Range to obstacle 1
robs2=sqrt((xo2-x)^2+(yo2-y)^2);           % Range to obstacle 2
diff1=atan(r1/robs1);
diff2=atan(r2/robs2);
if (theta >= (thetobs1-diff1)) & (theta <= (thetobs1+diff1))
    rho=robs1-r1;
elseif (theta >= (thetobs2-diff2)) & (theta <= (thetobs2+diff2))
    rho=robs2-r2;
else
    rho=500;                               % Assign large value if not on obstacle
end
rhon=rho*(1+0.01*randn(1,1));              % Add Range Error
thetan=theta+(0.45*pi/180)*randn(1,1);% Add Bearing Error
s=[rhon*cos(thetan);rhon*sin(thetan)];% Convert to x,y Components
end

```

```

%%%%%%      MATLAB Function VGOPEN.M calculates the value of the
%%%%%%%%      potential function and its gradient for a given input
%%%%%%%%      position and the parameter lamda for the open environment
%%%%%%%%      scenario including two obstacles.

```

```
%%%%%%%%
```

```
function [v,dvx,dvy]=vgopen(x,y,lamda);  
    global xo1 yo1 r1 xo2 yo2 r2          % Obstacle description  
    a=xo1;  
    b=yo1;  
    c=r1^2;  
    d=xo2;  
    e=yo2;  
    f=r2^2;  
    v0=((x-a).^2+(y-b).^2-c).*((x-d).^2+(y-e).^2-f);  
    z=v0/lamda;  
    v=sigmoid(z); % Apply "Squashing" Function  
    dvx=-dsig(z)*((2*(x-a)).*((x-d).^2+(y-e).^2-f)+((x-a).^2+(y-b).^2-c).*(2*(x-d)))/lamda;  
    % x-component of gradient  
    dvy=-dsig(z)*((2*(y-b)).*((x-d).^2+(y-e).^2-f)+((x-a).^2+(y-b).^2-c).*(2*(y-e)))/lamda;  
    % y-component of gradient  
end
```

```
%%%%%%%%    MATLAB File MODELSIM.m is the main program which implements the  
%%%%%%%%    navigation algorithm for a complex scenario where the environment  
%%%%%%%%    is comprised of multiple components.  
%%%%%%%%
```

```
% Prompt user for simulation parameters  
lamda=input('Enter value of lamda: ');  
mu=input('Enter value of mu: ');  
xhat0=input('Enter initial estimate of x-coordinate (ft): ');
```

```

yhat0=input('Enter initial estimate of y-coordinate (ft): ');
disp(' ');
% Initialize algorithm
phat0=[xhat0;yhat0]; % Initial Position Estimate
phat=zeros(2,Tf/9*400); % Initialize Output Position Matrix
theta=0; % Initial Sonar Bearing
phat(:,1)=phat0;
s0=sonmodel(state(1,1),state(2,1),theta); % Initial Sonar Vector
temps=phat0+[s0(1);s0(2)];
border(:,1)=temps; % Initial Pool Border Estimate
% Run Simulation
for n=2:k
    x=state(1,n); % Actual x-coordinate of AUV
    y=state(2,n); % Actual y-coordinate of AUV
    s=sonmodel(x,y,theta); % Noise-corrupted Sonar Vector
    z0=[state(3,n);state(4,n)*pi/180]; % Vehicle Velocity & Heading
    zcurr=z0+[randn(1)*0.05*z0(1);randn(1)*0.5*pi/180]; % w/ Error
    pdot=[zcurr(1)*cos(zcurr(2));zcurr(1)*sin(zcurr(2))]; % vx,vy
    shat=phat(:,n-1)+pdot*Ts+[s(1);s(2)]; % Estimated Position + Sonar Vector
    border(:,n)=shat; % Estimated Border Point of Environment
    temps=shat;
    % Calculate Value and Gradient of Potential Function
    xindex=round(shat(1)/0.1)+40;
    yindex=round(shat(2)/0.1)+40;
    if xindex >= 281
        xindex=281;
    end
    if yindex >= 281

```

```

    yindex=281;
end
if xindex <=1
    xindex=1;
end
if yindex <=1
    yindex=1;
end
% vval=v(xindex,yindex);
% dvxval=dvx(xindex,yindex);
% dvyval=dvy(xindex,yindex);
[vval,dvxval,dvyval]=vgmodel(shat(1),shat(2),lamda);
% Apply Correction Factor to Update x and y velocity estimates
pdotthat=pdot+mu*vval*[dvxval*sin(s(3));dvyval*cos(s(3))];
% Obtain current position estimate
phat(:,n)=phat(:,n-1)+Ts*pdotthat;
theta=theta+0.9*pi/180;    % Increment Sonar Head Bearing
if theta >=2*pi            % Reset theta to 0 after 360 deg
    theta=0;
end
end

%%%%%      MATLAB Function SONMODEL.m calculates the sonar range
%%%%%      and bearing to the complex environment consisting of
%%%%%      multiple components given the position and orientation of
%%%%%      the sonar head mounted on the vehicle.
%%%%%

```

```

function s=sonmodel(x,y,theta)
% Calculate bearing to each intersection of model components
theta1=atan2(16-y,20-x);
theta2=atan2(20-y,16-x);
if y > 10
    theta3=atan2(10-y,5-x)+2*pi;
else
    theta3=atan2(10-y,5-x);
end
theta4=atan2(y,x)+pi;
theta5=atan2(20-x,y)+3*pi/2;
rob3=sqrt((5-x)^2+(10-y)^2);
diff3=atan(0.2/rob3);
% Calculate range to component sonar beam is hitting
if theta <= theta1
    rho=(20-x)/cos(theta);
    sangle=pi/2;
elseif theta <=theta2
    rho=(20-x)/cos(theta)-(((20-x)*tan(theta)-(16-y))*sin(pi/4))/sin(pi/4+theta);
    sangle=pi/4;
elseif (theta >= (theta3-diff3)) & (theta <= (theta3+diff3))
    rho=rob3-0.2;
    sangle=pi/4;
elseif (theta >= theta4) & (theta <= theta5)
    rho=y/abs(cos(1.5*pi-theta));
    sangle=0;
elseif theta >= theta5
    rho=(20-x)/cos(theta);

```

```

    sangle=pi/2;
else
    rho=50;
    sangle=0;
end
rhon=rho*(1+0.0225*randn(1,1));    % Range + error
thetan=theta+(0.45*pi/180)*randn(1,1);    % Bearing + error
s=[rhon*cos(thetan);rhon*sin(thetan);sangle];    % Sonar return
end

```

```

%%%%%%%%    MATLAB Function VGMODEL.m calculates the value of the
%%%%%%%%    potential function and its gradient for the complex
%%%%%%%%    environment scenario.
%%%%%%%%

```

```

function [v,dvx,dvy]=vgmodel(x,y,lamda)
v0=y.*(x-20).*((y-16)+(x-16)-4).*((x-5).^2+(y-10).^2-0.2^2);
z=v0/lamda;
v=sigmoid(z);
dvx=-dsig(z)*(((y-16)+(x-16)-4).*((x-5).^2+(y-10).^2-0.2^2)+y.*(x-20).*((x-5).^2+
(y-10).^2-0.2^2)+(2*(x-5)).*y.*(x-20).*((y-16)+(x-16)-4))/lamda;
dvy=-dsig(z)*((x-20).*((y-16)+(x-16)-4).*((x-5).^2+(y-10).^2-0.2^2)+y.*(x-20).*
((x-5).^2+(y-10).^2-0.2^2)+(2*(y-10)).*y.*(x-20).*((y-16)+(x-16)-4))/lamda;
end

```



## E. PROGRAM LISTINGS FOR SIMULATIONS IN CHAPTER V

```
%%%%%%%% MATLAB script file SLIDEOBS.m implements a sliding mode
%%%%%%%% observer for a simple one-dimensional scenario as a proof
%%%%%%%% of concept.
%%%%%%%%
```

```
x0=2; % Initial actual position
k1=2; % Sliding mode gains
k2=4;
x1hat=3; % Initial position estimate
x2hat=0.0; % Initial velocity estimate
v=0.2; % Actual velocity
Ts=0.0225; % Integration step size
Tf=36; % Length of simulation
k=(Tf/Ts)+1; % Index
xhat=zeros(2,k); % Initialize estimate vector
xhat(:,1)=[x1hat;x2hat]; % Initial estimate x v
xact=zeros(2,k); % Initial state vector
xact(:,1)=[x0;v]; % Initial conditions
time=zeros(1,k); % Initialize time vector

% Iterative Algorithm
for i=2:k
    rhon=(12-xact(1,i-1))*(1+0.0225*randn(1,1)); % Sonar range w/ error
    vn=v*(1+0.0225*randn(1,1)); % Velocity measurement
    % Error and rate calculations
    x1tilda=xhat(1,i-1)-(12-rhon);
    x2tilda=xhat(2,i-1)-vn;
```

```

x1tildot=x2tilda-k1*sign(x1tilda);
x2tildot=-k2*sign(x1tilda);
x1hatdot=vn+x1tildot;
x2hatdot=x2tildot;
% Update position estimate
xhat(:,i)=xhat(:,i-1)+[x1hatdot;x2hatdot]*Ts;
% Udate actual position and time
xact(:,i)=[xact(1,i-1)+v*Ts,v];
time(i)=time(i-1)+Ts;
end

%%%%%%%% MATLAB script file SMOBS.m implements the sliding mode
%%%%%%%% observer for the closed border rectangular environment
%%%%%%%% scenario. It calls the function SONARSM.m to generate
%%%%%%%% the sonar data. Vehicle data is generated with file
%%%%%%%% AUVDATA.m.
%%%%%%%%

% Generate vehicle trajectory in component directions
xact=[state(1,:);state(3,:).*cos(state(4,:)*pi/180)];
yact=[state(2,:);state(3,:).*sin(state(4,:)*pi/180)];
% User input of initial state estimates
x1hat=input('Enter Initial Estimate of x-coordinate: ');
y1hat=input('Enter Initial Estimate of y-coordinate: ');
x2hat=input('Enter Initial x-component Velocity Estimate: ');
y2hat=input('Enter Initial y-component Velocity Estimate: ');
disp(' ');
Ts=0.0225; % Integration step size

```

```

k=length(state(1,:));           % Data length
k1=2;                           % Observer gains
k2=4;
phat0=[x1hat;x2hat;y1hat;y2hat]; % Initial estimates
phat=zeros(4,k);                % Initialize estimate vector
border=zeros(2,k-1);            % Initialize border vector
phat(:,1)=phat0;                % Initial conditions
theta=0;                        % Initial sonar bearing
% Iterative Algorithm
for n=2:k
    s=sonarsm(xact(1,n-1),yact(1,n-1),theta); % Sonar return w/ error
    flag=s(3);
    pdot=[xact(2,n-1)*(1+0.0225*randn(1));yact(2,n-1)*(1+0.0225*randn(1))];
    shat=[phat(1,n-1);phat(3,n-1)]+[s(1);s(2)]; % Tip of sonar vector
    border(:,n-1)=shat;          % Estimated position of border
    % Update x-coordinate components - return on right side of pool
    if flag == 1
        x1tilda=phat(1,n-1)-(20-s(1));
        x2tilda=phat(2,n-1)-pdot(1);
        x1tildot=x2tilda-k1*sign(x1tilda);
        x2tildot=-k2*sign(x1tilda);
        x1hatdot=pdot(1)+x1tildot;
        x2hatdot=x2tildot;
        xhat(:,n)=phat(1:2,n-1)+[x1hatdot;x2hatdot]*Ts;
        % Update state estimates
        phat(:,n)=[xhat(:,n);phat(3,n-1)+phat(4,n-1)*Ts;phat(4,n-1)];
    end
    % Update y-coordinate components - return on top side of pool

```

```

if flag == 2
    y1tilda=phat(3,n-1)-(20-s(2));
    y2tilda=phat(4,n-1)-pdot(2);
    y1tildot=y2tilda-k1*sign(y1tilda);
    y2tildot=-k2*sign(y1tilda);
    y1hatdot=pdot(2)+y1tildot;
    y2hatdot=y2tildot;
    yhat(:,n)=phat(3:4,n-1)+[y1hatdot;y2hatdot]*Ts;
    % Update state estimates
    phat(:,n)=[phat(1,n-1)+phat(2,n-1)*Ts;phat(2,n-1);yhat(:,n)];
end
% Update x-coordinate components - return on left side of pool
if flag == 3
    x1tilda=phat(1,n-1)-abs(s(1));
    x2tilda=phat(2,n-1)-pdot(1);
    x1tildot=x2tilda-k1*sign(x1tilda);
    x2tildot=-k2*sign(x1tilda);
    x1hatdot=pdot(1)+x1tildot;
    x2hatdot=x2tildot;
    xhat(:,n)=phat(1:2,n-1)+[x1hatdot;x2hatdot]*Ts;
    % Update state estimates
    phat(:,n)=[xhat(:,n);phat(3,n-1)+phat(4,n-1)*Ts;phat(4,n-1)];
end
% Update y-coordinate components - return on bottom side of pool
if flag == 4
    y1tilda=phat(3,n-1)-abs(s(2));
    y2tilda=phat(4,n-1)-pdot(2);
    y1tildot=y2tilda-k1*sign(y1tilda);

```

```

y2tildot=-k2*sign(y1tilda);
y1hatdot=pdot(2)+y1tildot;
y2hatdot=y2tildot;
yhat(:,n)=phat(3:4,n-1)+[y1hatdot;y2hatdot]*Ts;
% Update state estimates
phat(:,n)=[phat(1,n-1)+phat(2,n-1)*Ts;phat(2,n-1);yhat(:,n)];
end
theta=theta+0.9*pi/180;    % Increment sonar bearing
if theta >=2*pi            % Reset to zero after 360 deg scan
    theta=0;
end
end

```

```

%%%%%%%% MATLAB Function SONARSM.m calculates the sonar range and
%%%%%%%% bearing to the square pool border given the position in
%%%%%%%% the pool and the angle of the sonar head. The bearing
%%%%%%%% and range are corrupted with noise (error). A flag is returned
%%%%%%%% to the calling program to indicated which side of the pool
%%%%%%%% the sonar beam is hitting.
%%%%%%%%

```

```

function s = sonarsm(x,y,theta);
% Calculate angles from position to 4 pool corners
theta1=atan2(20-y,20-x);    % Upper Right Corner
theta2=atan2(x,20-y)+pi/2;   % Upper Left Corner
theta3=atan2(y,x)+pi;        % Lower Left Corner
theta4=atan2(20-x,y)+3*(pi/2); % Lower Right Corner
if theta <= theta1

```

```

    rho=(20-x)/cos(theta);
    flag=1;
elseif theta <= theta2
    rho=(20-y)/sin(theta);
    flag=2;
elseif theta <= theta3
    rho=x/cos(pi-theta);
    flag=3;
elseif theta <= theta4
    rho=y/abs(cos(1.5*pi-theta));
    flag=4;
else
    rho=(20-x)/cos(theta);
    flag=1;
end
rhon=rho*(1+0.0225*randn(1,1));      % Add Range Error
thetan=theta+(0.45*pi/180)*randn(1,1); % Add Bearing Error
s=[rhon*cos(thetan);rhon*sin(thetan);flag]; % Convert to x,y Components
end

```

#### **F. AUV SIMULINK MODEL**

In addition to the MATLAB script file AUVDATA.m the SIMULINK model AUVMODEL.m can be utilized to generate the complete state trajectory of the AUV for any simulation. This model can account for the motion resulting from two commanded maneuvers during the vehicle run. The model is presented as Figure C.1.

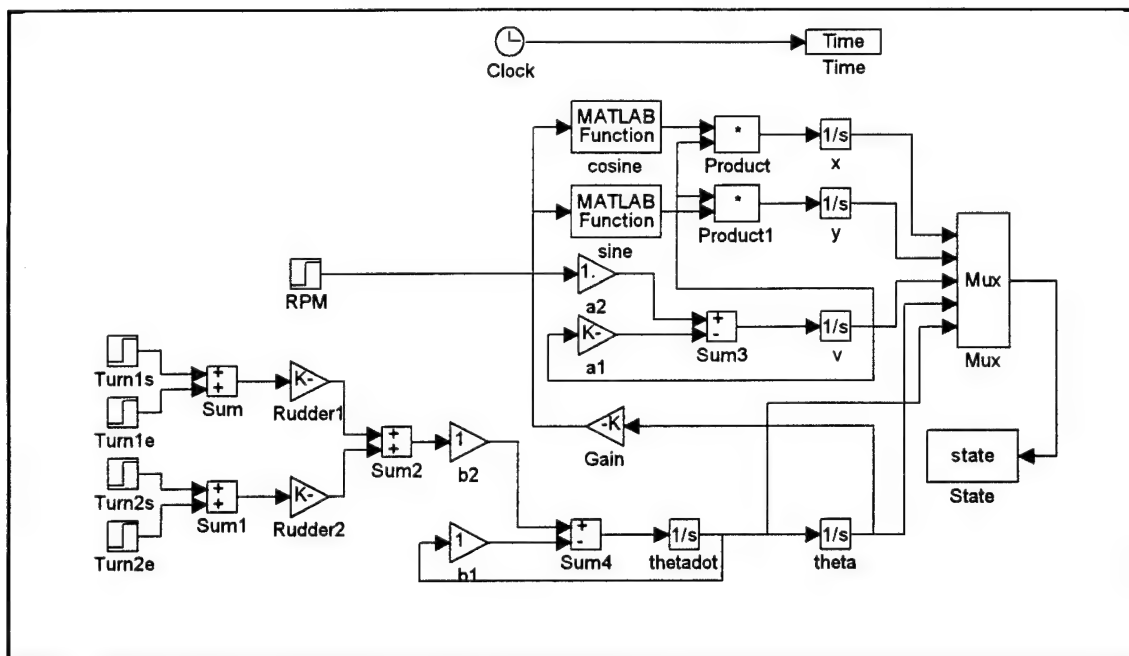


Figure C.1 SIMULINK Model AUVMODEL.m





## APPENDIX D. SMOOTHING FILTER DESIGN

The design of digital filters is extensively covered in the technical literature. The choice of a smoothing filter for the purpose of this thesis follows the principles outlined in [Ref. 8].

A logical choice in determining a strategy for filtering (or smoothing) the AUV estimated position data is to base the filter output on an average of the parameter data measured over the previous sonar scan (i.e. 400 data points). If we consider  $x(n)$  as the input data sequence and  $y(n)$  as the output of the filter the following difference equation is obtained:

$$y(n) = \frac{1}{400}(x(n) + \dots + x(n - 400 + 1)) \quad (D.1)$$

Taking the z-transform of this difference equation yields

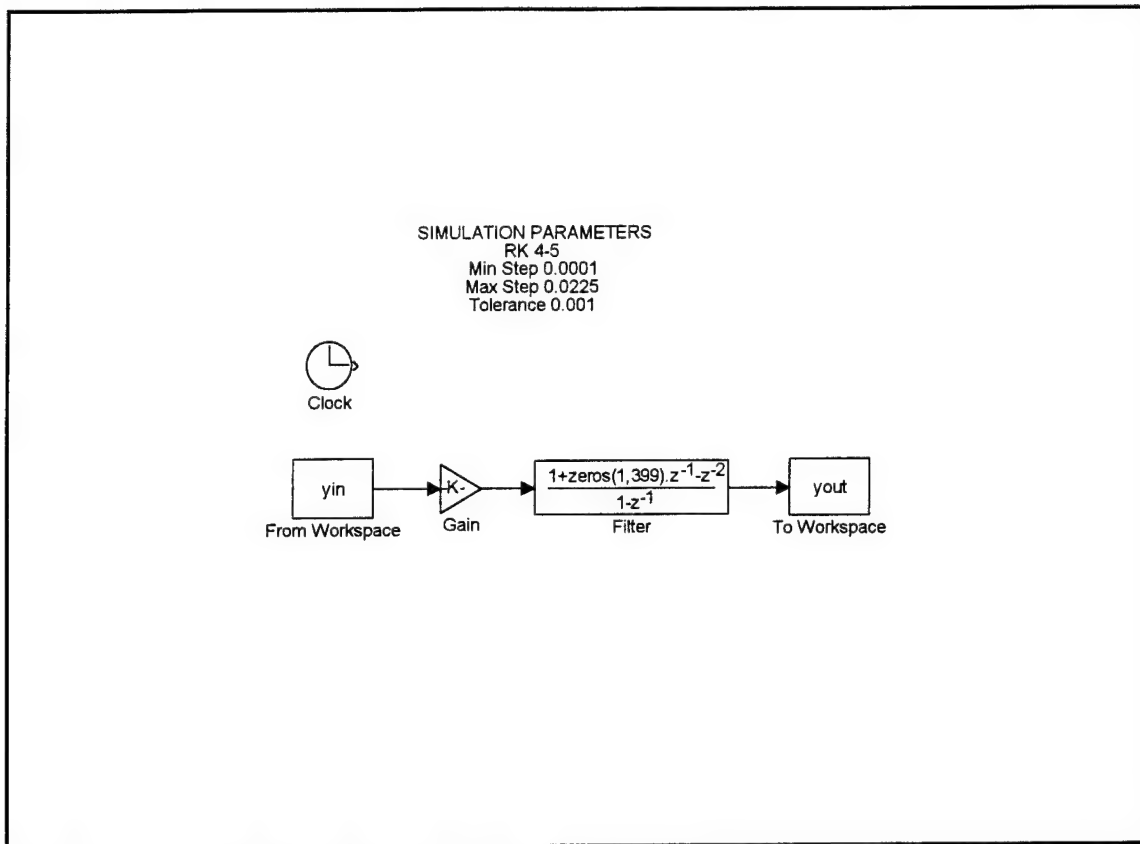
$$H(z) = \frac{1}{400} \sum_{k=0}^{399} z^{-k} = \frac{1 - z^{-400}}{1 - z^{-1}} \quad (D.2)$$

Now, application of the inverse z-transform to the simplified system function of Equation (D.2) yields

$$y(n) = y(n - 1) + \frac{1}{400}(x(n) - x(n - 400)) \quad (D.3)$$

which simply states that the filter output is the previous output plus a weighted sum of the change in data over the previous sonar scan. The SIMULINK Model of the filter is presented as Figure D.1 and is followed by the MATLAB script file FILTXY.m which formats the input and commands operation of the model. The filter is initialized with a vector of 400

components equal to the initial position estimate. An accurate filtered output value is not available until the first sonar scan is completed (400 measurements).



**Figure D.1 SIMULINK Model of Smoothing Filter**

```
%%%%%%%% MATLAB File FILTXY.M filters the x and y estimates of
%%%%%%%% AUV position utilizing a SIMULINK Model AUVFILT.M.
%%%%%%%%
```

```
xsize=size(phat);           % Determine data length
long=xsize(2);
yinx=[xhat0*ones(1,400) phat(1,:)]; % Initialize filter input x
yiny=[yhat0*ones(1,400) phat(2,:)]; % Initialize filter input y
```

```

realt=0:Ts:Tf;           % Real time vector
tindex=0:Ts:Tf+9;        % Data input time vector
yin=[tindex' yinx'];     % Filter input x-coordinate
dummy=rk45('auvfilt',[0 Tf+9],[ ],[Ts,Ts,1e-3]); % Run filter
clear dummy
filtx=yout(600:long+399); % Generate filtered x output
clear yout                % Clear model output
yin=[tindex' yiny'];     % Filter input y-coordinate
dummy=rk45('auvfilt',[0 Tf+9],[ ],[Ts,Ts,1e-3]); % Run filter
clear dummy
filty=yout(600:long+399); % Generate filtered y output

```



## LIST OF REFERENCES

1. Various, *OCEANS '93 Engineering in Harmony with the Ocean Proceedings*, Oceanic Engineering Society of the Institute of Electrical and Electronics Engineers, Inc., October 1993.
2. R. Cristi, M. Caccia, G. Veruggio, A. J. Healy, " A Sonar Based Approach to AUV Localization," paper presented at the Third Workshop on Controls Applications in Marine Systems (CAMS '95), Trondheim, Norway, May 1995.
3. E. Percin, *Sonar Localization of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1993.
4. A. Kayirhan, *Sonar Based Navigation of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1994.
5. J.-J. E. Slotine, J. K. Hedrick, E. A. Misawa, " On Sliding Observers for Nonlinear Systems," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 109, p.245, September 1987.
6. M. Santina, A. Stubberud, and G. Hostetter, *Digital Control System Design*, Saunders College Publishing, Orlando, FL, 1994.
7. R. Hippenstiel, "Parameter Estimation," Notes for EC 3310 (Linear Optimal Estimation and Control), Naval Postgraduate School, Monterey, CA, 1994 (Unpublished).
8. J. Proakis, and G. Manolakis, *Digital Signal Processing*, Macmillan Publishing Company, New York, NY, 1988.



## INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
4.	Professor R. Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
5.	Professor R. Hutchins, Code EC/Hu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
6.	Professor H. Titus, Code EC/Ts Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5121	1
7.	LCDR Kevin D. Conowitch, USN 43621 21st St. W. Lancaster, CA 93535	2